PROGRAMACIÓN II

Trabajo Práctico 5: Relaciones UML 1 a 1

1. Pasaporte - Foto - Titular
   a. Composición: **Pasaporte → Foto**
   b. Asociación bidireccional: **Pasaporte ↔ Titular**

   Clases y atributos:

   i. Pasaporte: numero, fechaEmision
   ii. Foto: imagen, formato
   iii. Titular: nombre, dni

```java
/*1. Pasaporte - Foto - Titular
        a. Composición: Pasaporte → Foto
        b. Asociación bidireccional: Pasaporte ↔ Titular
        Clases y atributos:
        i. Pasaporte: numero, fechaEmision
        ii. Foto: imagen, formato
        iii. Titular: nombre, dni
 */
import java.time.LocalDate;

class Foto {  2 usages
    private byte[] imagen;  1 usage
    private String formato;  1 usage
    @Contract(pure = true)
    public Foto(byte[] imagen, String formato) {  no usages
        this.imagen = imagen; this.formato = formato;
    }
}
class Titular {  2 usages
    private String nombre;  1 usage
    private String dni;  1 usage
    private Pasaporte pasaporte; // extremo bidireccional  1 usage

    @Contract(pure = true)
    public Titular(String nombre, String dni) {  no usages
        this.nombre = nombre; this.dni = dni;
    }
    public void setPasaporte(Pasaporte pasaporte) { this.pasaporte = pasaporte; }
}
```

```java
public class Pasaporte {  2 usages
    private String numero;  1 usage
    private LocalDate fechaEmision;  1 usage
    private final Foto foto;      // composición  1 usage
    private Titular titular;      // extremo bidireccional  1 usage

    @Contract(value = "_, _, null -> fail", pure = true)
    public Pasaporte(String numero, LocalDate fechaEmision, Foto foto) {  no usages
        if (foto == null) throw new IllegalArgumentException("Foto obligatoria");
        this.numero = numero; this.fechaEmision = fechaEmision; this.foto = foto;
    }
    public void setTitular(Titular titular) { this.titular = titular; }  no usages
}
```

2. Celular - Batería - Usuario
    a. Agregación: **Celular → Batería**
    b. Asociación bidireccional: **Celular ↔ Usuario**

Clases y atributos:

i. Celular: imei, marca, modelo
ii. Batería: modelo, capacidad
iii. Usuario: nombre, dni

```
class Bateria {  2 usages
    private String modelo;  1 usage
    private int capacidad;  1 usage

    @Contract(pure = true)
    public Bateria(String modelo, int capacidad) {  no usages
        this.modelo = modelo;
        this.capacidad = capacidad;
    }
}
class Usuario {  2 usages
    private String nombre;  1 usage
    private String dni;  1 usage
    private Celular celular; // extremo bidi  1 usage

    @Contract(pure = true)
    public Usuario(String nombre, String dni) {  no usages
        this.nombre = nombre;
        this.dni = dni;
    }
    public void setCelular(Celular celular) {  no usages
        this.celular = celular;
    }
}

public class Celular {  2 usages
    private String imei;  1 usage
    private String marca;  1 usage
    private String modelo;  1 usage

    private Bateria bateria;  1 usage

    private Usuario usuario;  1 usage

    @Contract(pure = true)
    public Celular(String imei, String marca, String modelo) {
        this.imei = imei;
        this.marca = marca;
        this.modelo = modelo;
    }

    public void setBateria(Bateria bateria) {  no usages
        this.bateria = bateria;
    }

    public void setUsuario(Usuario usuario) {  no usages
        this.usuario = usuario;
    }
}
```

3. Libro - Autor - Editorial
    a. Asociación unidireccional: **Libro → Autor**
    b. Agregación: **Libro → Editorial**

Clases y atributos:

i. Libro: titulo, isbn
ii. Autor: nombre, nacionalidad
iii. Editorial: nombre, dirección

```
public class Libro {  no usages
    class Autor {  2 usages
        private String nombre; private String nacionalidad;  1 usage
        @Contract(pure = true)
        public Autor(String nombre, String nacionalidad) { this.nombre = nombre; this.nacionalidad = nacionalidad; }
    }

    class Editorial {  2 usages
        private String nombre; private String direccion;  1 usage
        @Contract(pure = true)
        public Editorial(String nombre, String direccion) { this.nombre = nombre; this.direccion = direccion; }  no usa
    }

    class Libro {  no usages
        private String titulo; private String isbn;  1 usage
        private Autor autor; // unidireccional  1 usage
        private Editorial editorial; // agregación  1 usage

        @Contract(pure = true)
        public Libro(String titulo, String isbn, Autor autor) {  no usages
            this.titulo = titulo; this.isbn = isbn; this.autor = autor;
        }
        public void setEditorial(Editorial editorial) { this.editorial = editorial; }  no usages
    }
}
```

4. TarjetaDeCrédito - Cliente - Banco
    a. Asociación bidireccional: **TarjetaDeCrédito ↔ Cliente**
    b. Agregación: **TarjetaDeCrédito → Banco**

Clases y atributos:

i. TarjetaDeCrédito: numero, fechaVencimiento
ii. Cliente: nombre, dni
iii. Banco: nombre, cuit

```
import java.time.LocalDate;

class Banco { private String nombre, cuit;  2 usages
    public Banco(String n, String c){  no usages
        this.nombre=n; this.cuit=c; }
}


class Cliente {  2 usages

    private String nombre, dni;  1 usage
    private TarjetaDeCredito tarjeta;  1 usage
    public Cliente(String n,String d){  no usages
        this.nombre=n; this.dni=d;
    }
    public void setTarjeta(TarjetaDeCredito t){  no usages
        this.tarjeta=t; }
}


class TarjetaDeCredito {  2 usages
    private String numero; private LocalDate fechaVencimiento;  1 usage
    private Cliente cliente; // bidi  1 usage
    private Banco banco; // agregación  1 usage


    @Contract(pure = true)
    public TarjetaDeCredito(String numero, LocalDate fechaVencimiento) {
        this.numero = numero; this.fechaVencimiento = fechaVencimiento;
    }
    public void setCliente(Cliente cliente){ this.cliente = cliente; }
    public void setBanco(Banco banco){ this.banco = banco; }  no usages
}
```

5. Computadora - PlacaMadre - Propietario
    a. Composición: **Computadora → PlacaMadre**
    b. Asociación bidireccional: **Computadora ↔ Propietario**

Clases y atributos:

i. Computadora: marca, numeroSerie
ii. PlacaMadre: modelo, chipset
iii. Propietario: nombre, dni

```
class PlacaMadre {  2 usages

    private String modelo, chipset;  1 usage
    public PlacaMadre(String m,String c){  no usages
        this.modelo=m;
        this.chipset=c; }
}

class Propietario {  2 usages
    private String nombre, dni;  1 usage
    private Computadora computadora;  1 usage
    public Propietario(String n,String d){  no usages
        this.nombre=n; this.dni=d;
    }
    public void setComputadora(Computadora c){  no usages
        this.computadora=c; }
}
class Computadora {  2 usages
    private String marca, numeroSerie;  1 usage
    private final PlacaMadre placaMadre;  1 usage
    private Propietario propietario;  1 usage
    @Contract("_, _, null -> fail")
    public Computadora(String marca, String numeroSerie, PlacaMadre placaMadre) {  no usages
        if (placaMadre==null) throw new IllegalArgumentException("Placa madre requerida");
        this.marca=marca; this.numeroSerie=numeroSerie;
        this.placaMadre=placaMadre;
    }
    public void setPropietario(Propietario propietario){  no usages
        this.propietario = propietario;
    }
}
```

6. Reserva - Cliente - Mesa
    a. Asociación unidireccional: **Reserva → Cliente**
    b. Agregación: **Reserva → Mesa**

Clases y atributos:

i. Reserva: fecha, hora
ii. Cliente: nombre, telefono
iii. Mesa: numero, capacidad

```
import java.time.*;

    class ClienteR {  2 usages

        private String nombre, telefono;  1 usage
        public ClienteR(String n,String t){  no usages
            this.nombre=n; this.telefono=t; }
    }


    class Mesa {  2 usages

        private int numero, capacidad;  1 usage
        public Mesa(int n,int c){  no usages
            this.numero=n;
            this.capacidad=c; }
    }
    class Reserva {  no usages
        private LocalDate fecha;  1 usage
        private LocalTime hora;  1 usage
        private ClienteR cliente;  1 usage
        private Mesa mesa;  1 usage
        @Contract(pure = true)
        public Reserva(LocalDate fecha, LocalTime hora, ClienteR cliente){  no usages
            this.fecha=fecha;
            this.hora=hora;
            this.cliente=cliente;
        }
        public void setMesa(Mesa mesa){  no usages
            this.mesa = mesa;
        }
    }
```

7. Vehículo - Motor - Conductor
    a. Agregación: **Vehículo → Motor**
    b. Asociación bidireccional: **Vehículo ↔ Conductor**

Clases y atributos:

i. Vehículo: patente, modelo
ii. Motor: tipo, numeroSerie
iii. Conductor: nombre, licencia

```
class Motor {  2 usages

    private String tipo, numeroSerie;  1 usage
    public Motor(String t,String n){  no usages
        this.tipo=t; this.numeroSerie=n; }
}


class Conductor { private String nombre, licencia;  2 usages
    private Vehiculo vehiculo; public Conductor(String n,String l){  1 usage
        this.nombre=n;
        this.licencia=l;
    }
    public void setVehiculo(Vehiculo v){  no usages
        this.vehiculo=v; }
}
class Vehiculo {  2 usages

    private String patente, modelo;  1 usage
    private Motor motor;  1 usage
    private Conductor conductor;  1 usage
    public Vehiculo(String p,String m){  no usages
        this.patente=p; this.modelo=m;
    }
    public void setMotor(Motor motor){  no usages
        this.motor=motor;
    }
    public void setConductor(Conductor c){  no usages
        this.conductor=c;
    }
}
```

8. Documento - FirmaDigital - Usuario
    a. Composición: **Documento → FirmaDigital**
    b. Agregación: **FirmaDigital → Usuario**

    Clases y atributos:

        i. Documento: titulo, contenido
        ii. FirmaDigital: codigoHash, fecha
        iii. Usuario: nombre, email

```
import java.time.LocalDateTime;

class UsuarioFD {  2 usages

        private String nombre, email;  1 usage
        public UsuarioFD(String n,String e){  no usages
            this.nombre=n; this.email=e; }
    }

    class FirmaDigital {  2 usages
        private String codigoHash;  1 usage
        private LocalDateTime fecha;  1 usage
        private UsuarioFD usuario; // agregación  1 usage
        @Contract(pure = true)
        public FirmaDigital(String codigoHash, LocalDateTime fecha, UsuarioFD usuario){  no usag
            this.codigoHash=codigoHash;
            this.fecha=fecha;
            this.usuario=usuario;
        }
    }
    class Documento {  no usages

        private String titulo, contenido;  1 usage
        private final FirmaDigital firma;  1 usage
        @Contract("_, _, null -> fail")
        public Documento(String t,String c,FirmaDigital f){  no usages
            if (f==null) throw new IllegalArgumentException("Firma requerida");
            this.titulo=t; this.contenido=c;
            this.firma=f; }
    }
```

9. CitaMédica - Paciente - Profesional
    a. Asociación unidireccional: **CitaMédica → Paciente,**
    b. Asociación unidirecciona: **CitaMédica → Profesional**

Clases y atributos:

i. CitaMédica: fecha, hora
ii. Paciente: nombre, obraSocial
iii. Profesional: nombre, especialidad

```java
import java.time.*;

class Paciente {  2 usages

    private String nombre, obraSocial;  1 usage
    public Paciente(String n,String o){  no usages
        this.nombre=n; this.obraSocial=o; }
}


class Profesional {  2 usages

    private String nombre, especialidad;  1 usage
    public Profesional(String n,String e){  no usages
        this.nombre=n; this.especialidad=e; }
}

class CitaMedica {  no usages

    private LocalDate fecha;  1 usage
    private LocalTime hora; private Paciente paciente;  1 usage
    private Profesional profesional;  1 usage
    @Contract(pure = true)
    public CitaMedica(LocalDate f, LocalTime h, Paciente p, Profesional pr){  no
        this.fecha=f; this.hora=h; this.paciente=p; this.profesional=pr; }
}
```

10. CuentaBancaria - ClaveSeguridad - Titular
    a. Composición: **CuentaBancaria → ClaveSeguridad**
    b. Asociación bidireccional: **CuentaBancaria ↔ Titular**

Clases y atributos:

i. CuentaBancaria: cbu, saldo
ii. ClaveSeguridad: codigo, ultimaModificacion
iii. Titular: nombre, dni.

```
import java.time.*;

    class ClaveSeguridad {  2 usages
        private String codigo;  1 usage
        private LocalDateTime ultimaModificacion;  1 usage
        @Contract(pure = true)
        public ClaveSeguridad(String c, LocalDateTime u){  no usages
            this.codigo=c; this.ultimaModificacion=u; }
    }
    class TitularCB {  2 usages
        private String nombre, dni;  1 usage
        private CuentaBancaria cuenta;  1 usage
        public TitularCB(String n,String d){  no usages
            this.nombre=n; this.dni=d;
        }
        public void setCuenta(CuentaBancaria c){  no usages
            this.cuenta=c; }
    }
    class CuentaBancaria {  2 usages
        private String cbu;  1 usage
        private double saldo;  1 usage
        private final ClaveSeguridad clave;  1 usage
        private TitularCB titular;  1 usage
        @Contract(value = "_, _, null -> fail", pure = true)
        public CuentaBancaria(String cbu, double saldo, ClaveSeguridad clave){  no usage
            if(clave==null) throw new IllegalArgumentException("Clave requerida");
            this.cbu=cbu;
            this.saldo=saldo;
            this.clave=clave;
        }
        public void setTitular(TitularCB t){  no usages
            this.titular=t; }
    }
```

DEPENDENCIA DE USO

La clase usa otra como **parámetro de un método**, pero **no la guarda como atributo**.

Ejercicios de Dependencia de Uso

11. Reproductor - Canción - Artista
    a. Asociación unidireccional: **Canción → Artista**
    b. Dependencia de uso: **Reproductor.reproducir(Cancion)**

Clases y atributos:

i. Canción: titulo.
ii. Artista: nombre, genero.
iii. Reproductor->método: void reproducir(Cancion cancion)

```
class Artista {  2 usages
private String nombre, genero;  1 usage
public Artista(String n,String g){  no usages
    this.nombre=n; this.genero=g; }
}
class Cancion {  1 usage
private String titulo;  1 usage
private Artista artista;  1 usage
@Contract(pure = true)
public Cancion(String t, Artista a){  no usages
    this.titulo=t;
    this.artista=a; }
}
class Reproductor {  no usages
    public void reproducir(Cancion cancion) {  no usages
        if (cancion == null) throw new IllegalArgumentException("Canción requerida");

        System.out.println("Reproduciendo canción…");
    }
}
```

12. Impuesto - Contribuyente - Calculadora
    a. Asociación unidireccional: **Impuesto → Contribuyente**
    b. Dependencia de uso: **Calculadora.calcular(Impuesto)**

Clases y atributos:

i. Impuesto: monto.
ii. Contribuyente: nombre, cuil.
iii. Calculadora->método: void calcular(Impuesto impuesto)

```
class Contribuyente {  2 usages
private String nombre, cuil;  1 usage
public Contribuyente(String n,String c){  no usages
    this.nombre=n;
    this.cuil=c; }
}
class Impuesto {  1 usage
private double monto;  1 usage
private Contribuyente contribuyente;  1 usage
@Contract(pure = true)
public Impuesto(double m, Contribuyente c){  no usages
    this.monto=m;
    this.contribuyente=c; }
}
class Calculadora {  no usages
public void calcular(Impuesto impuesto){  no usages
    if(impuesto==null) throw new IllegalArgumentException("Impuesto requerido");
}
```

DEPENDENCIA DE CREACIÓN

La clase crea otra dentro de un método, pero no la conserva como atributo..

Ejercicios de Dependencia de Creación

13. GeneradorQR - Usuario - CódigoQR
    a. Asociación unidireccional: **CódigoQR → Usuario**
    b. Dependencia de creación: **GeneradorQR.generar(String, Usuario)**

Clases y atributos:

i. CodigoQR: valor.
ii. Usuario: nombre, email.
iii. GeneradorQR->método: void generar(String valor, Usuario usuario)

```
class UsuarioQR {  3 usages
private String nombre, email;  1 usage
public UsuarioQR(String n,String e){  no usages
    this.nombre=n;
    this.email=e; }
}
class CodigoQR {  2 usages
private String valor;  1 usage
private UsuarioQR usuario;  1 usage
@Contract(pure = true)
public CodigoQR(String v, UsuarioQR u){  1 usage
    this.valor=v;
    this.usuario=u; }
}
class GeneradorQR {  no usages
public void generar(String valor, UsuarioQR usuario) { CodigoQR qr = new CodigoQR(valor, usuario); }
```

14. EditorVideo - Proyecto - Render
    a. Asociación unidireccional: Render → Proyecto
    b. Dependencia de creación: **EditorVideo.exportar(String, Proyecto)**
    c. Clases y atributos:
        i. Render: formato.
        ii. Proyecto: nombre, duracionMin.
        iii. EditorVideo->método: void exportar(String formato, Proyecto proyecto)

```
class Proyecto {  3 usages
private String nombre;  1 usage
private int duracionMin;  1 usage
@Contract(pure = true)
public Proyecto(String n,int d){  no usages
    this.nombre=n;
    this.duracionMin=d;
}
}
class Render {  2 usages
private String formato;  1 usage
private Proyecto proyecto;  1 usage
@Contract(pure = true)
public Render(String f, Proyecto p){  1 usage
    this.formato=f;
    this.proyecto=p;
}
}
class EditorVideo {  no usages
public void exportar(String formato, Proyecto proyecto) { Render r = new Render(formato, proyecto); }
}
```

Link Repositorio Github https://github.com/fabian24cf/Programacion2.git