Trabajo Práctico 5: Relaciones UML 1 a 1

Caso Práctico

Desarrollar los siguientes ejercicios en Java. Cada uno deberá incluir:

- Diagrama UML
- Tipo de relación (asociación, agregación, composición, dependencia)
- Dirección (unidireccional o bidireccional)
- Implementación de las clases con atributos y relaciones definidas

Ejercicios de Relaciones 1 a 1

1. Pasaporte - Foto - Titular a. Composición: Pasaporte → Foto

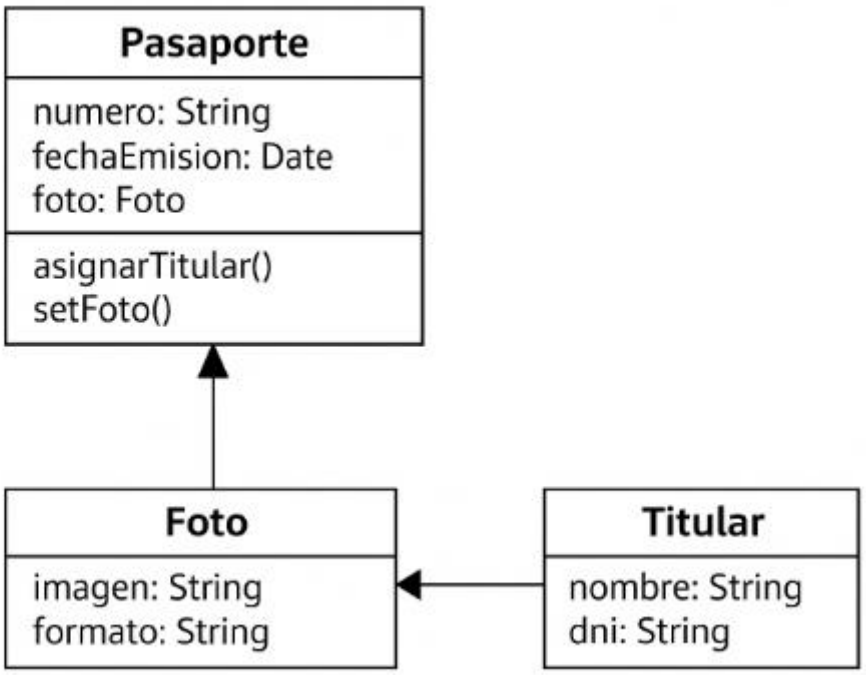
b. Asociación bidireccional: Pasaporte ↔ Titular

Clases y atributos:

i. Pasaporte: numero, fechaEmision

ii. Foto: imagen, formato

iii. Titular: nombre, dni



1. Análisis de Relaciones

Clases involucradas:

- Pasaporte
- Foto
- Titular

Relaciones:

Relación	**Tipo**	**Dirección**	**Descripción**
Pasaporte → Foto	**Composición (1 a 1)**	**Unidireccional**	El pasaporte contiene una foto. Si el pasaporte se elimina, la foto también.
Pasaporte ↔ Titular	**Asociación (1 a 1)**	**Bidireccional**	El pasaporte pertenece a un titular, y el titular tiene un pasaporte.

Implementación en Java

Foto.java

```
public class Foto {
    private String imagen;
    private String formato;

    public Foto(String imagen, String formato) {
        this.imagen = imagen;
        this.formato = formato;
    }

    public String getImagen() {
        return imagen;
    }

    public String getFormato() {
        return formato;
    }

    @Override
    public String toString() {
        return "Foto [imagen=" + imagen + ", formato=" + formato + "]";
    }
}
```

Titular.java

```
public class Titular {
    private String nombre;
    private String dni;
```

```
    private Pasaporte pasaporte; // Asociación bidireccional

    public Titular(String nombre, String dni) {
        this.nombre = nombre;
        this.dni = dni;
    }

    public String getNombre() {
        return nombre;
    }

    public String getDni() {
        return dni;
    }

    public Pasaporte getPasaporte() {
        return pasaporte;
    }

    public void setPasaporte(Pasaporte pasaporte) {
        this.pasaporte = pasaporte;
    }

    @Override
    public String toString() {
        return "Titular [nombre=" + nombre + ", dni=" + dni + "]";
    }
}
```

Pasaporte.java

```
import java.util.Date;

public class Pasaporte {
    private String numero;
    private Date fechaEmision;
    private Foto foto; // Composición
    private Titular titular; // Asociación bidireccional

    public Pasaporte(String numero, Date fechaEmision, Foto foto) {
        this.numero = numero;
        this.fechaEmision = fechaEmision;
        this.foto = foto;
    }

    public void asignarTitular(Titular titular) {
        this.titular = titular;
        titular.setPasaporte(this); // enlace bidireccional
    }

    public String getNumero() {
        return numero;
    }

    public Date getFechaEmision() {
        return fechaEmision;
    }

    public Foto getFoto() {
```

```
        return foto;
    }

    public Titular getTitular() {
        return titular;
    }

    @Override
    public String toString() {
        return "Pasaporte [numero=" + numero + ", fechaEmision=" +
fechaEmision +
                ", foto=" + foto + ", titular=" + titular.getNombre() + "]";
    }
}
```

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

Análisis de Relaciones

Clases involucradas:

- Celular
- Batería

- Usuario

Relaciones:

Relación	Tipo	Dirección	Descripción
Celular → Batería	**Agregación (1 a 1)**	**Unidireccional**	El celular tiene una batería, pero puede existir sin ella (no dependencia total).
Celular ↔ Usuario	**Asociación (1 a 1)**	**Bidireccional**	El usuario posee un celular y el celular pertenece a un usuario.

Bateria.java

```
public class Bateria {
    private String modelo;
    private int capacidad; // en mAh

    public Bateria(String modelo, int capacidad) {
        this.modelo = modelo;
        this.capacidad = capacidad;
    }

    public String getModelo() {
        return modelo;
    }

    public int getCapacidad() {
        return capacidad;
    }

    @Override
    public String toString() {
```

```
        return "Batería [modelo=" + modelo + ", capacidad=" + capacidad +
"mAh]";
    }
}
```

Usuario.java

```
public class Usuario {
    private String nombre;
    private String dni;
    private Celular celular; // Asociación bidireccional

    public Usuario(String nombre, String dni) {
        this.nombre = nombre;
        this.dni = dni;
    }

    public String getNombre() {
        return nombre;
    }

    public String getDni() {
        return dni;
    }

    public Celular getCelular() {
        return celular;
    }

    public void setCelular(Celular celular) {
```

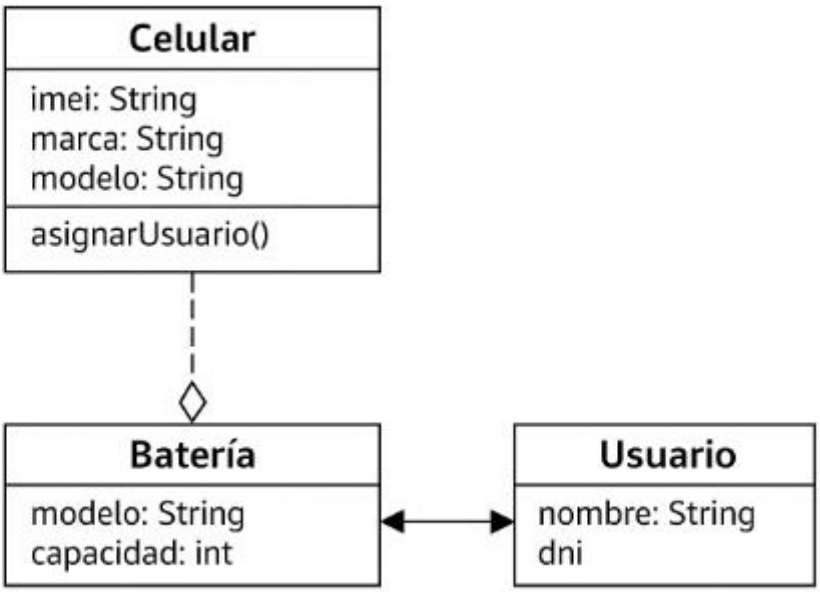
```
        this.celular = celular;
    }

    @Override
    public String toString() {
        return "Usuario [nombre=" + nombre + ", dni=" + dni + "]";
    }
}
```

Celular.java

```
public class Celular {
    private String imei;
    private String marca;
    private String modelo;
    private Bateria bateria; // Agregación
    private Usuario usuario; // Asociación bidireccional

    public Celular(String imei, String marca, String modelo, Bateria bateria) {
        this.imei = imei;
        this.marca = marca;
        this.modelo = modelo;
        this.bateria = bateria;
    }

    public void asignarUsuario(Usuario usuario) {
        this.usuario = usuario;
        usuario.setCelular(this); // relación bidireccional
    }
```

```
    public String getImei() {
        return imei;
    }

    public String getMarca() {
        return marca;
    }

    public String getModelo() {
        return modelo;
    }

    public Bateria getBateria() {
        return bateria;
    }

    public Usuario getUsuario() {
        return usuario;
    }

    @Override
    public String toString() {
        return "Celular [IMEI=" + imei + ", marca=" + marca + ", modelo=" +
modelo +
                ", batería=" + bateria + ", usuario=" + usuario.getNombre() + "]";
    }
}
```

Diagrama UML



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, direccion

Análisis de Relaciones

Clases involucradas:

- Libro
- Autor
- Editorial

Relaciones:

Relación	Tipo	Dirección	Descripción
Libro → Autor	**Asociación (1 a 1)**	**Unidireccional**	El libro conoce a su autor, pero el autor no conoce directamente al libro.
Libro → Editorial	**Agregación (1 a 1)**	**Unidireccional**	El libro pertenece a una editorial, pero la editorial puede existir sin el libro.

Autor.java

```
public class Autor {
    private String nombre;
    private String nacionalidad;

    public Autor(String nombre, String nacionalidad) {
        this.nombre = nombre;
        this.nacionalidad = nacionalidad;
    }

    public String getNombre() {
        return nombre;
    }

    public String getNacionalidad() {
        return nacionalidad;
    }

    @Override
    public String toString() {
        return "Autor [nombre=" + nombre + ", nacionalidad=" + nacionalidad + "]";
    }
```

```
}
```

Editorial.java

```
public class Editorial {
    private String nombre;
    private String direccion;

    public Editorial(String nombre, String direccion) {
        this.nombre = nombre;
        this.direccion = direccion;
    }

    public String getNombre() {
        return nombre;
    }

    public String getDireccion() {
        return direccion;
    }

    @Override
    public String toString() {
        return "Editorial [nombre=" + nombre + ", direccion=" + direccion + "]";
    }
}
```

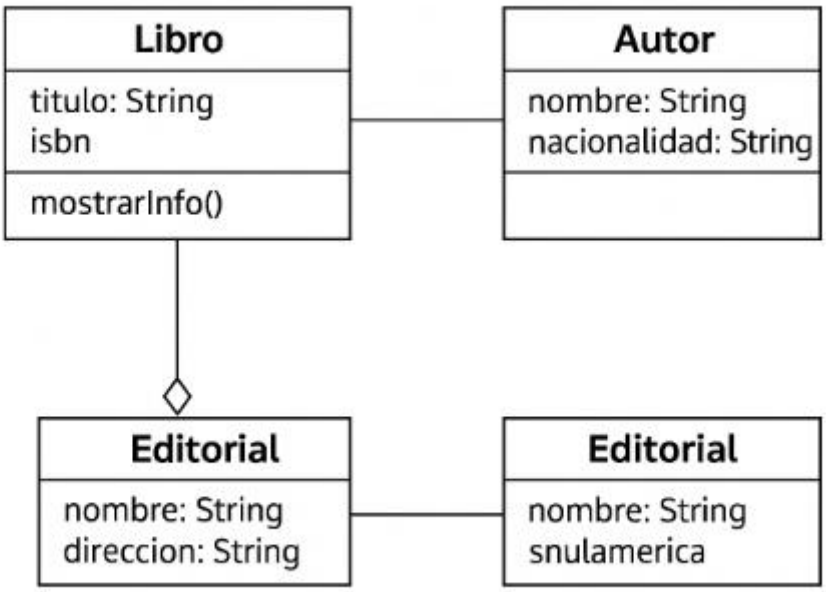
Libro.java

```
public class Libro {
    private String titulo;
```

```
private String isbn;
private Autor autor;        // Asociación unidireccional
private Editorial editorial; // Agregación

public Libro(String titulo, String isbn, Autor autor, Editorial editorial) {
    this.titulo = titulo;
    this.isbn = isbn;
    this.autor = autor;
    this.editorial = editorial;
}

public String getTitulo() {
    return titulo;
}

public String getIsbn() {
    return isbn;
}

public Autor getAutor() {
    return autor;
}

public Editorial getEditorial() {
    return editorial;
}

public void mostrarInfo() {
    System.out.println("Libro: " + titulo + " (ISBN: " + isbn + ")");
```

```
        System.out.println("Autor: " + autor.getNombre() + " - " +
autor.getNacionalidad());
        System.out.println("Editorial: " + editorial.getNombre() + " - " +
editorial.getDireccion());
    }

    @Override
    public String toString() {
        return "Libro [titulo=" + titulo + ", isbn=" + isbn +
                ", autor=" + autor.getNombre() + ", editorial=" + editorial.getNombre()
+ "]";
    }
}
```

Diagrama UML



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

Relaciones

Asociación bidireccional: TarjetaDeCredito ↔ Cliente

Agregación: TarjetaDeCredito → Banco

Banco.java

```
public class Banco {
    private String nombre;
    private String cuit;
```

```
    public Banco(String nombre, String cuit) {
        this.nombre = nombre;
        this.cuit = cuit;
    }

    public String getNombre() {
        return nombre;
    }

    public String getCuit() {
        return cuit;
    }

    @Override
    public String toString() {
        return "Banco: " + nombre + " (CUIT: " + cuit + ")";
    }
}
```

Cliente.java

```
public class Cliente {
    private String nombre;
    private String dni;
    private TarjetaDeCredito tarjeta; // Asociación bidireccional

    public Cliente(String nombre, String dni) {
        this.nombre = nombre;
        this.dni = dni;
```

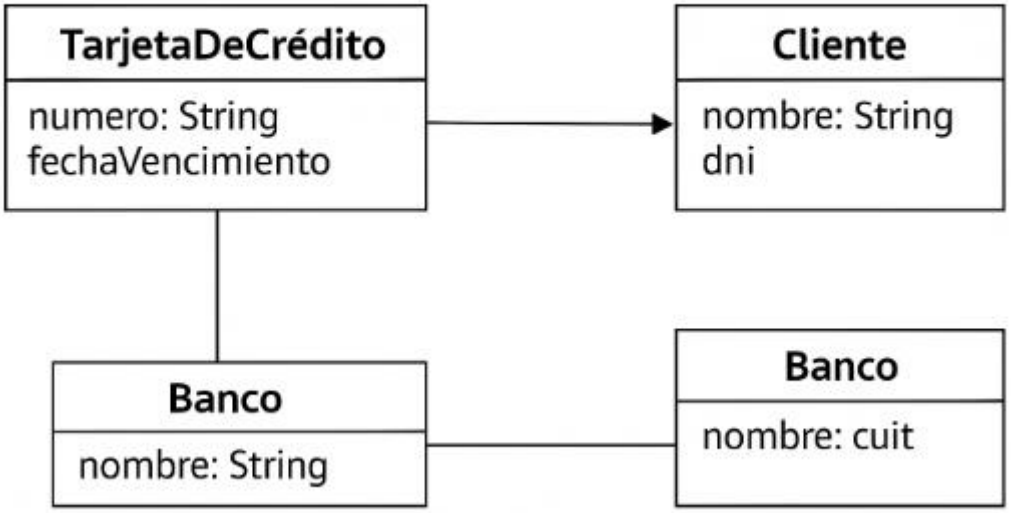
```
    }

    public void setTarjeta(TarjetaDeCredito tarjeta) {
        this.tarjeta = tarjeta;
    }

    public TarjetaDeCredito getTarjeta() {
        return tarjeta;
    }

    @Override
    public String toString() {
        return "Cliente: " + nombre + " (DNI: " + dni + ")";
    }
}
```

TarjetaDeCredito.java

```
public class TarjetaDeCredito {
    private String numero;
    private String fechaVencimiento;
    private Cliente cliente; // Asociación bidireccional
    private Banco banco;     // Agregación

    public TarjetaDeCredito(String numero, String fechaVencimiento, Banco
banco) {
        this.numero = numero;
        this.fechaVencimiento = fechaVencimiento;
        this.banco = banco;
    }
```

```
    public void setCliente(Cliente cliente) {
        this.cliente = cliente;
        cliente.setTarjeta(this); // vinculación bidireccional
    }

    public Cliente getCliente() {
        return cliente;
    }

    public Banco getBanco() {
        return banco;
    }

    @Override
    public String toString() {
        return "TarjetaDeCredito Nº: " + numero +
                ", Vence: " + fechaVencimiento +
                ", Banco: " + banco.getNombre();
    }
}
```

Diagrama UML



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

Relaciones

Relación	**Tipo**	**Dirección**	**Descripción**
Computadora → PlacaMadre	**Composición (1 a 1)**	**Unidireccional**	La PlacaMadre no puede existir sin la Computadora.

Relación	Tipo	Dirección	Descripción
Computadora ↔ Propietario	**Asociación bidireccional (1 a 1)**	**Bidireccional**	La computadora tiene un propietario, y el propietario conoce su computadora.

PlacaMadre.java

```
public class PlacaMadre {
    private String modelo;
    private String chipset;

    public PlacaMadre(String modelo, String chipset) {
        this.modelo = modelo;
        this.chipset = chipset;
    }

    public String getModelo() {
        return modelo;
    }

    public String getChipset() {
        return chipset;
    }

    @Override
    public String toString() {
        return "PlacaMadre [modelo=" + modelo + ", chipset=" + chipset + "]";
    }
}
```

Propietario.java

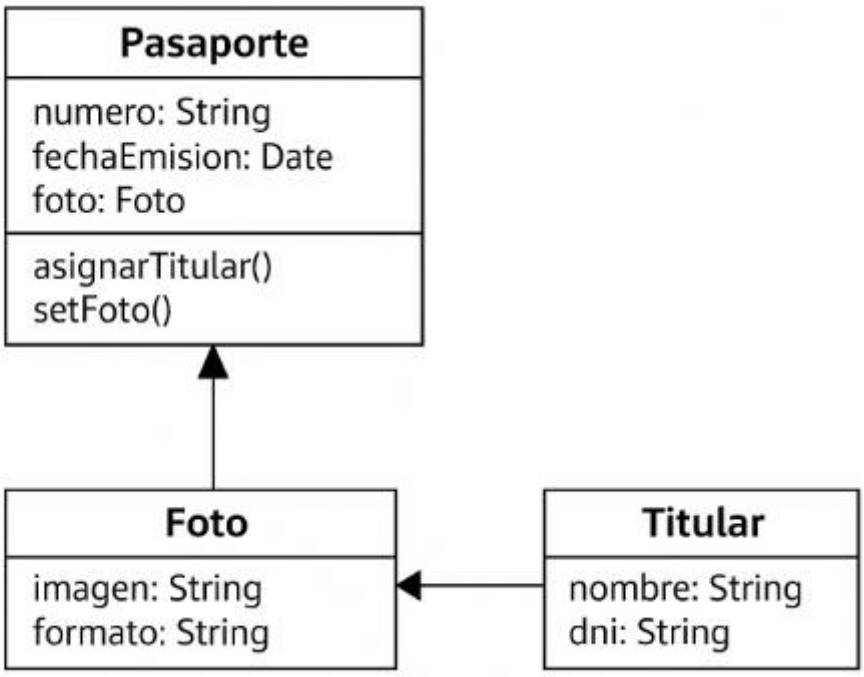
```
public class Propietario {
    private String nombre;
    private String dni;
    private Computadora computadora; // Asociación bidireccional

    public Propietario(String nombre, String dni) {
        this.nombre = nombre;
        this.dni = dni;
    }

    public void setComputadora(Computadora computadora) {
        this.computadora = computadora;
    }

    public Computadora getComputadora() {
        return computadora;
    }

    @Override
    public String toString() {
        return "Propietario [nombre=" + nombre + ", dni=" + dni + "]";
    }
}
```

Computadora.java

```
public class Computadora {
    private String marca;
    private String numeroSerie;
    private PlacaMadre placaMadre;   // Composición
    private Propietario propietario; // Asociación bidireccional
```

```
    public Computadora(String marca, String numeroSerie, String modeloPlaca,
String chipsetPlaca) {
        this.marca = marca;
        this.numeroSerie = numeroSerie;
        this.placaMadre = new PlacaMadre(modeloPlaca, chipsetPlaca); //
composición → creada dentro
    }

    public void setPropietario(Propietario propietario) {
        this.propietario = propietario;
        propietario.setComputadora(this); // vinculación bidireccional
    }

    public void mostrarInfo() {
        System.out.println("Computadora marca: " + marca + " (N° Serie: " +
numeroSerie + ")");
        System.out.println("Placa madre: " + placaMadre.getModelo() + " -
Chipset: " + placaMadre.getChipset());
        if (propietario != null) {
            System.out.println("Propietario: " + propietario);
        }
    }

    @Override
    public String toString() {
        return "Computadora [marca=" + marca + ", numeroSerie=" + numeroSerie
+ "]";
    }
}
```

Diagrama UML



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

Relaciones

Relación	**Tipo**	**Dirección**	**Descripción**
Reserva → Cliente	**Asociación (1 a 1)**	**Unidireccional**	La reserva conoce al cliente, pero el cliente no conoce a la reserva.
Reserva → Mesa	**Agregación (1 a 1)**	**Unidireccional**	La reserva está asociada a una mesa, pero la mesa puede existir sin la reserva.

Cliente.java

```
public class Cliente {
    private String nombre;
    private String telefono;

    public Cliente(String nombre, String telefono) {
        this.nombre = nombre;
        this.telefono = telefono;
    }

    public String getNombre() {
        return nombre;
    }

    public String getTelefono() {
        return telefono;
    }

    @Override
    public String toString() {
        return "Cliente [nombre=" + nombre + ", telefono=" + telefono + "]";
    }
}
```

Mesa.java

```
public class Mesa {
    private int numero;
```

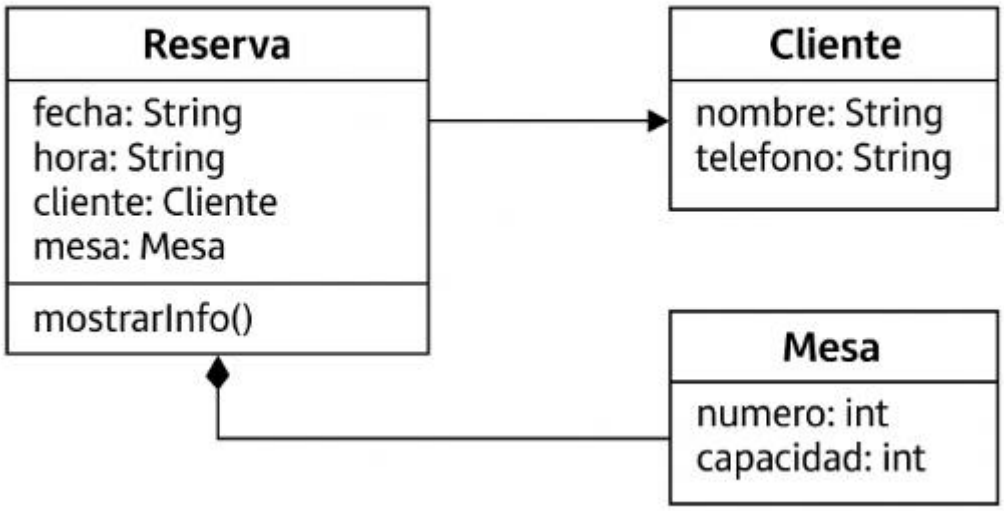
```
    private int capacidad;

    public Mesa(int numero, int capacidad) {
        this.numero = numero;
        this.capacidad = capacidad;
    }

    public int getNumero() {
        return numero;
    }

    public int getCapacidad() {
        return capacidad;
    }

    @Override
    public String toString() {
        return "Mesa [n°=" + numero + ", capacidad=" + capacidad + " personas]";
    }
}
```

Reserva.java

```
public class Reserva {
    private String fecha;
    private String hora;
    private Cliente cliente; // Asociación unidireccional
    private Mesa mesa;       // Agregación

    public Reserva(String fecha, String hora, Cliente cliente, Mesa mesa) {
        this.fecha = fecha;
```

```
        this.hora = hora;
        this.cliente = cliente;
        this.mesa = mesa;
    }

    public void mostrarInfo() {
        System.out.println("Reserva para " + cliente.getNombre());
        System.out.println("Fecha: " + fecha + " - Hora: " + hora);
        System.out.println("Mesa asignada: " + mesa);
    }

    @Override
    public String toString() {
        return "Reserva [fecha=" + fecha + ", hora=" + hora + ", cliente=" +
cliente.getNombre() + "]";
    }
}
```

Diagrama UML



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

Relaciones

Agregación: Vehiculo → Motor

Asociación bidireccional: Vehiculo ↔ Conductor

ClaseMotor.java

```
public class Motor {
    private String tipo;
    private String numeroSerie;

    public Motor(String tipo, String numeroSerie) {
```

```
        this.tipo = tipo;
        this.numeroSerie = numeroSerie;
    }

    public String getTipo() {
        return tipo;
    }

    public String getNumeroSerie() {
        return numeroSerie;
    }

    @Override
    public String toString() {
        return "Motor [tipo=" + tipo + ", número de serie=" + numeroSerie +
"]";
    }
}
```

ClaseConductor.java

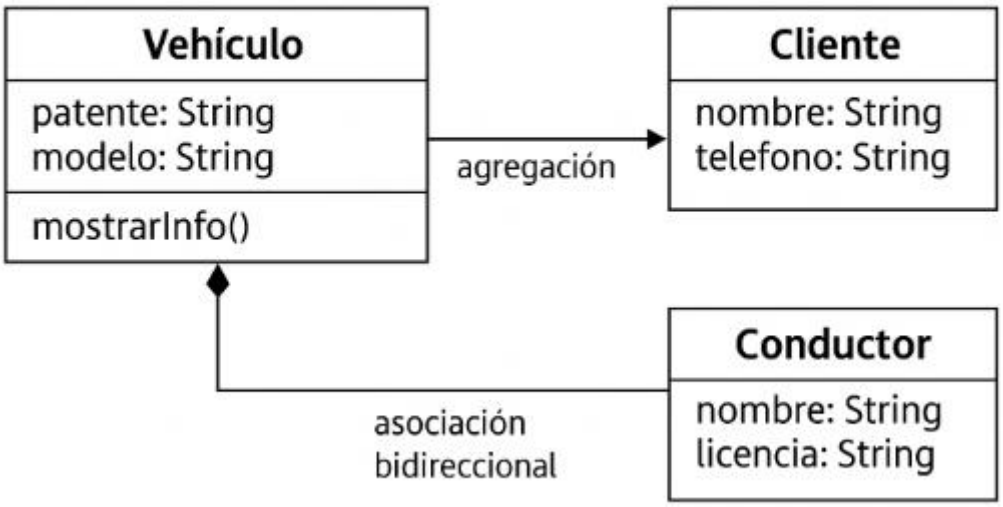
```
public class Conductor {

    private String nombre;
    private String licencia;
    private Vehiculo vehiculo; // Asociación bidireccional

    public Conductor(String nombre, String licencia) {
        this.nombre = nombre;
        this.licencia = licencia;
    }

    public void setVehiculo(Vehiculo vehiculo) {
        this.vehiculo = vehiculo;
    }

    public Vehiculo getVehiculo() {
```

```
        return vehiculo;
    }

    @Override

    public String toString() {
        return "Conductor [nombre=" + nombre + ", licencia=" + licencia + "]";

    }
}
```

Clasevehiculo.java

```
public class Vehiculo {

    private String patente;
    private String modelo;
    private Motor motor;       // Agregación
    private Conductor conductor; // Asociación bidireccional

    public Vehiculo(String patente, String modelo, Motor motor) {
        this.patente = patente;
        this.modelo = modelo;
        this.motor = motor;
    }

    public void setConductor(Conductor conductor) {
        this.conductor = conductor;
        conductor.setVehiculo(this); // Vinculación bidireccional
    }

    public Conductor getConductor() {
        return conductor;
    }

    public Motor getMotor() {
        return motor;
    }

    @Override
    public String toString() {
        return "Vehiculo [patente=" + patente + ", modelo=" + modelo + ", motor="
+ motor + "]";

    }
}
```

Diagrama UML



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

Relaciones

Composición: Documento → FirmaDigital (la firma digital pertenece solo a un documento)

Agregación: FirmaDigital → Usuario (la firma usa un usuario existente)

Claseusuario.java

```
public class Usuario {
    private String nombre;
    private String email;

    public Usuario(String nombre, String email) {
        this.nombre = nombre;
        this.email = email;
    }

    public String getNombre() {
        return nombre;
    }

    public String getEmail() {
        return email;
    }

    @Override
    public String toString() {
        return "Usuario [nombre=" + nombre + ", email=" + email + "]";
    }
}
```

FirmaDigital.java

```
public class FirmaDigital {
    private String codigoHash;
    private String fecha;
    private Usuario usuario; // Agregación
```

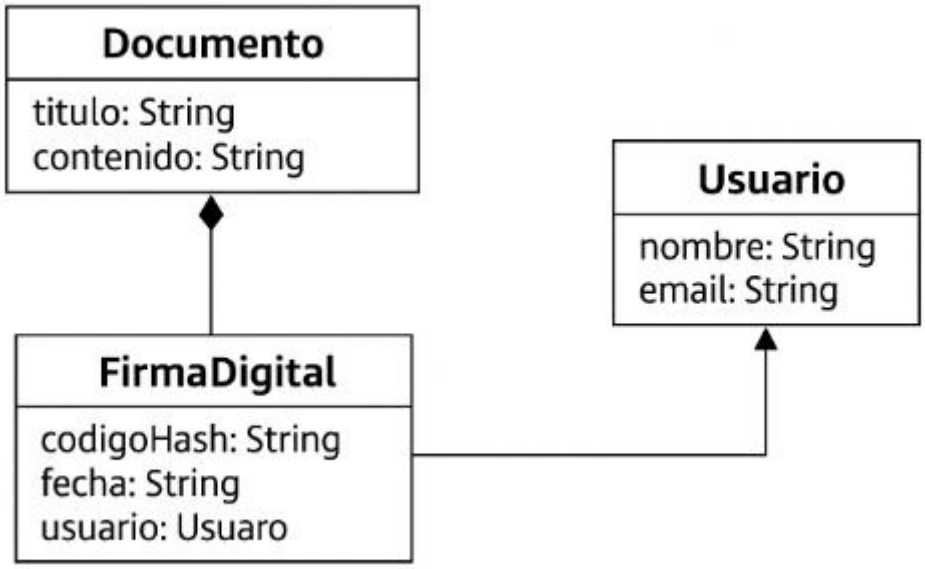
```
    public FirmaDigital(String codigoHash, String fecha, Usuario usuario) {
        this.codigoHash = codigoHash;
        this.fecha = fecha;
        this.usuario = usuario;
    }

    public String getCodigoHash() {
        return codigoHash;
    }

    public String getFecha() {
        return fecha;
    }

    public Usuario getUsuario() {
        return usuario;
    }

    @Override
    public String toString() {
        return "FirmaDigital [codigoHash=" + codigoHash + ", fecha=" + fecha + ",
usuario=" + usuario.getNombre() + "]";
    }
}
```

ClaseDocumento.java

```
public class Documento {
    private String titulo;
    private String contenido;
```

```
    private FirmaDigital firmaDigital; // Composición

    public Documento(String titulo, String contenido, Usuario usuario) {
        this.titulo = titulo;
        this.contenido = contenido;
        // la firma se crea dentro del documento → composición
        this.firmaDigital = new FirmaDigital(generarHash(), obtenerFechaActual(),
usuario);
    }

    private String generarHash() {
        return "HASH-" + Math.abs(contenido.hashCode());
    }

    private String obtenerFechaActual() {
        return java.time.LocalDate.now().toString();
    }

    public FirmaDigital getFirmaDigital() {
        return firmaDigital;
    }

    @Override
    public String toString() {
        return "Documento [titulo=" + titulo + ", contenido=" + contenido + ",
firma=" + firmaDigital + "]";
    }
}
```

Diagrama UML



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

Relaciones

Relación	**Tipo**	**Dirección**	**Descripción**
CitaMédica → Paciente	Asociación	Unidireccional	La cita conoce al paciente, pero el paciente no conoce la cita.
CitaMédica → Profesional	Asociación	Unidireccional	La cita conoce al profesional, pero el profesional no conoce la cita.

Paciente.java

public class Paciente {

```
    private String nombre;
    private String obraSocial;

    public Paciente(String nombre, String obraSocial) {
        this.nombre = nombre;
        this.obraSocial = obraSocial;
    }

    public String getNombre() {
        return nombre;
    }

    public String getObraSocial() {
        return obraSocial;
    }

    @Override
    public String toString() {
        return "Paciente [nombre=" + nombre + ", obra social=" + obraSocial + "]";
    }
}
```

Profesional.java

```
public class Profesional {
    private String nombre;
    private String especialidad;

    public Profesional(String nombre, String especialidad) {
        this.nombre = nombre;
```

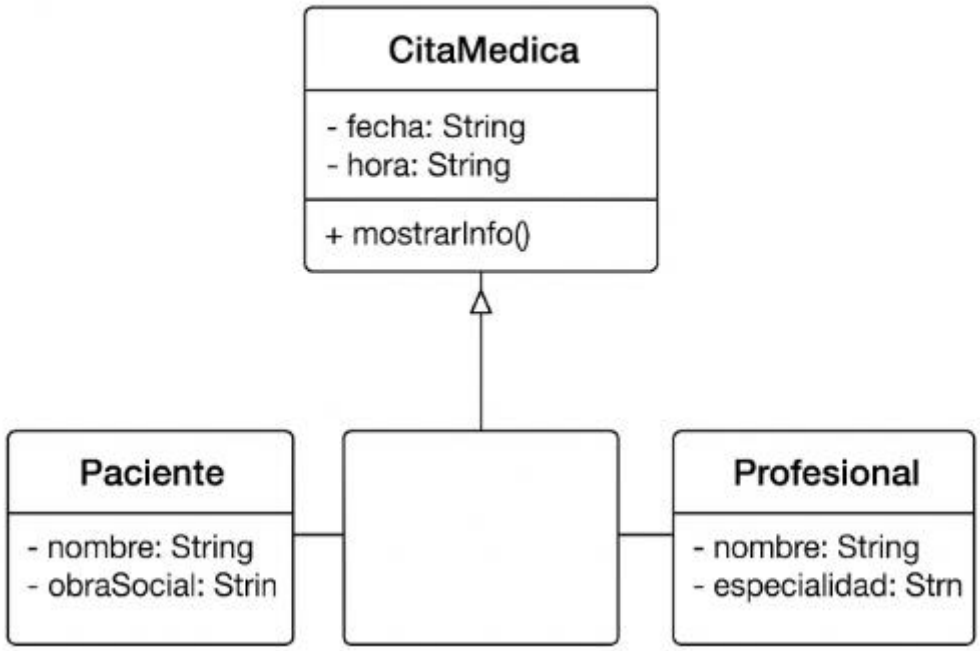
```
        this.especialidad = especialidad;
    }

    public String getNombre() {
        return nombre;
    }

    public String getEspecialidad() {
        return especialidad;
    }

    @Override
    public String toString() {
        return "Profesional [nombre=" + nombre + ", especialidad=" + especialidad
+ "]";
    }
}
```

Citamedica.java

```
public class CitaMedica {
    private String fecha;
    private String hora;
    private Paciente paciente;        // Asociación unidireccional
    private Profesional profesional;  // Asociación unidireccional

    public CitaMedica(String fecha, String hora, Paciente paciente, Profesional
profesional) {
        this.fecha = fecha;
        this.hora = hora;
        this.paciente = paciente;
```

```
        this.profesional = profesional;
    }

    public void mostrarInfo() {
        System.out.println("📅 Cita Médica");
        System.out.println("Fecha: " + fecha + " - Hora: " + hora);
        System.out.println("Paciente: " + paciente.getNombre() + " (" +
paciente.getObraSocial() + ")");
        System.out.println("Profesional: " + profesional.getNombre() + " - " +
profesional.getEspecialidad());
    }

    @Override
    public String toString() {
        return "CitaMedica [fecha=" + fecha + ", hora=" + hora + ", paciente=" +
paciente.getNombre() +
                ", profesional=" + profesional.getNombre() + "]";
    }
}
```

Diagrama UML



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.

Relaciones:

- Composición: CuentaBancaria → ClaveSeguridad
  (Si se elimina la cuenta, también se elimina la clave).
- Asociación bidireccional: CuentaBancaria ↔ Titular
  (Cada cuenta tiene un titular y el titular puede tener varias cuentas).

CuentaBancaria.java

```
public class CuentaBancaria {
    private String cbu;
    private double saldo;
    private ClaveSeguridad claveSeguridad;  // Composición
    private Titular titular;                // Asociación bidireccional

    public CuentaBancaria(String cbu, double saldo, Titular titular, String
codigoClave) {
        this.cbu = cbu;
        this.saldo = saldo;
        this.titular = titular;
        this.claveSeguridad = new ClaveSeguridad(codigoClave);
        titular.agregarCuenta(this); // vínculo bidireccional
    }

    public void mostrarInfo() {
        System.out.println("CBU: " + cbu + ", Saldo: $" + saldo);
        System.out.println("Titular: " + titular.getNombre());
        System.out.println("Clave: " + claveSeguridad.getCodigo());
    }
    // Getters y setters
    public String getCbu() { return cbu; }
    public double getSaldo() { return saldo; }
    public Titular getTitular() { return titular; }
}
```

ClaveSeguridad.java

```
import java.time.LocalDateTime;

public class ClaveSeguridad {
```

```
    private String codigo;
    private LocalDateTime ultimaModificacion;

    public ClaveSeguridad(String codigo) {
        this.codigo = codigo;
        this.ultimaModificacion = LocalDateTime.now();
    }

    public void actualizarClave(String nuevaClave) {
        this.codigo = nuevaClave;
        this.ultimaModificacion = LocalDateTime.now();
    }

    public String getCodigo() { return codigo; }
    public LocalDateTime getUltimaModificacion() { return ultimaModificacion; }
}
```

Titular.java

```
import java.util.ArrayList;
import java.util.List;

public class Titular {
    private String nombre;
    private String dni;
    private List<CuentaBancaria> cuentas; // Relación bidireccional

    public Titular(String nombre, String dni) {
        this.nombre = nombre;
        this.dni = dni;
        this.cuentas = new ArrayList<>();
```

```
    }

    public void agregarCuenta(CuentaBancaria cuenta) {
        if (!cuentas.contains(cuenta)) {
            cuentas.add(cuenta);
        }
    }

    public void mostrarCuentas() {
        System.out.println("Titular: " + nombre + " - Cuentas asociadas:");
        for (CuentaBancaria c : cuentas) {
            System.out.println("  • " + c.getCbu());
        }
    }

    public String getNombre() { return nombre; }
    public String getDni() { return dni; }
}
```

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.

Relaciones pedidas

1. **Composición:**
   CuentaBancaria **contiene** una ClaveSeguridad
   → Si se elimina la CuentaBancaria, también desaparece la ClaveSeguridad.
2. **Asociación bidireccional:**
   CuentaBancaria ↔ Titular
   → Ambos conocen la existencia del otro.

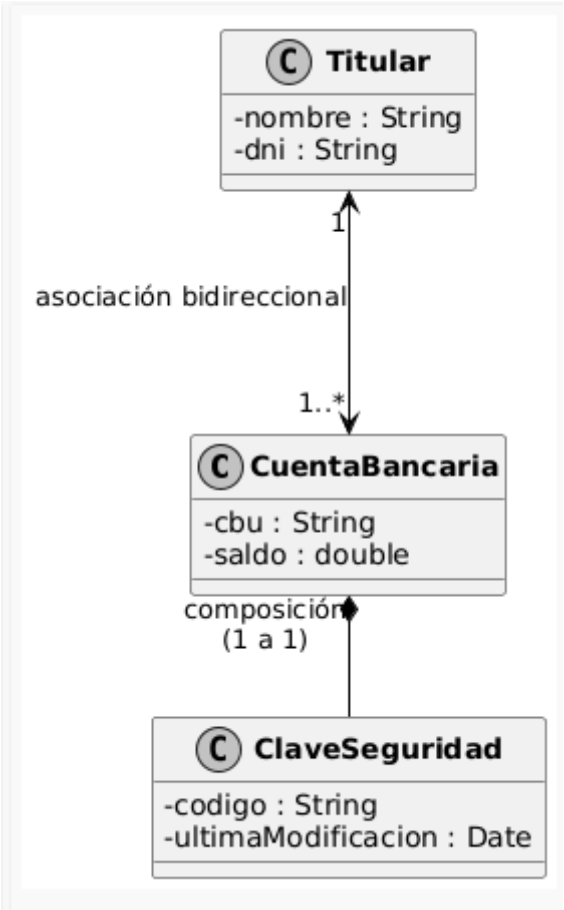
```
import java.util.Date;
import java.util.ArrayList;
import java.util.List;

// ----------------------
// Clase ClaveSeguridad
// ----------------------
public class ClaveSeguridad {
    private String codigo;
    private Date ultimaModificacion;

    public ClaveSeguridad(String codigo, Date ultimaModificacion) {
        this.codigo = codigo;
        this.ultimaModificacion = ultimaModificacion;
    }

    public String getCodigo() {
        return codigo;
    }

    public void setCodigo(String codigo) {
```

```
        this.codigo = codigo;
        this.ultimaModificacion = new Date(); // cada cambio actualiza la fecha
    }

    public Date getUltimaModificacion() {
        return ultimaModificacion;
    }

    @Override
    public String toString() {
        return "ClaveSeguridad{" +
                "codigo='" + codigo + '\'' +
                ", ultimaModificacion=" + ultimaModificacion +
                '}';
    }
}

// ----------------------
// Clase Titular
// ----------------------
public class Titular {
    private String nombre;
    private String dni;
    private List<CuentaBancaria> cuentas;  // relación bidireccional

    public Titular(String nombre, String dni) {
        this.nombre = nombre;
        this.dni = dni;
        this.cuentas = new ArrayList<>();
```

```
}

public String getNombre() {
    return nombre;
}

public String getDni() {
    return dni;
}

public List<CuentaBancaria> getCuentas() {
    return cuentas;
}

// Método para vincular cuenta ↔ titular
public void agregarCuenta(CuentaBancaria cuenta) {
    if (!cuentas.contains(cuenta)) {
        cuentas.add(cuenta);
        cuenta.setTitular(this);
    }
}

@Override
public String toString() {
    return "Titular{" +
            "nombre='" + nombre + '\'' +
            ", dni='" + dni + '\'' +
            '}';
}
```

}

Diagrama UML



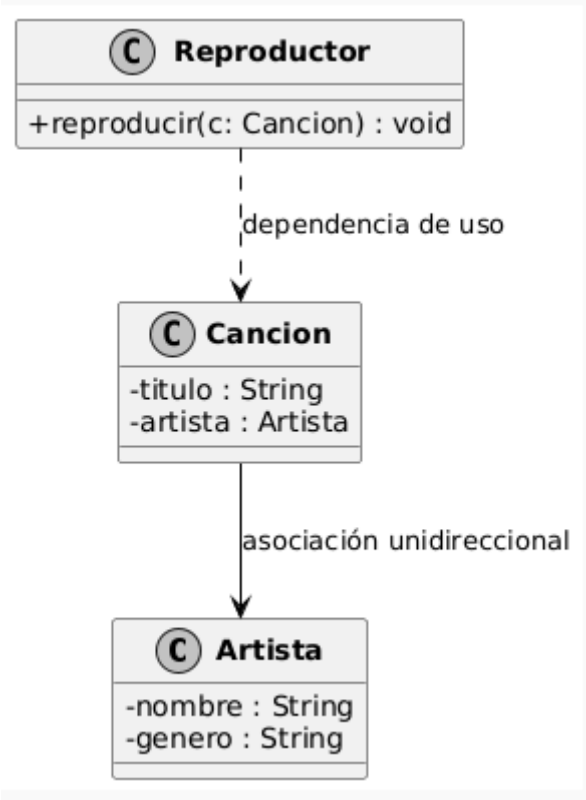
11. Ejercicios de Dependencia de Uso

Relaciones pedidas

1. Asociación unidireccional:
   Cancion → Artista
   - La clase Cancion *conoce* al Artista, pero el Artista no conoce a la Cancion.
2. Dependencia de uso:
   Reproductor.reproducir(Cancion)
   - Reproductor *usa* a Cancion como parámetro, pero no la guarda como atributo

```
// ----------------------
// Clase Artista
// ----------------------
public class Artista {
    private String nombre;
    private String genero;

    public Artista(String nombre, String genero) {
        this.nombre = nombre;
        this.genero = genero;
    }

    public String getNombre() {
        return nombre;
    }

    public String getGenero() {
        return genero;
    }

    @Override
    public String toString() {
        return nombre + " (" + genero + ")";
    }
}
```

Diagrama UML



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)

Relaciones

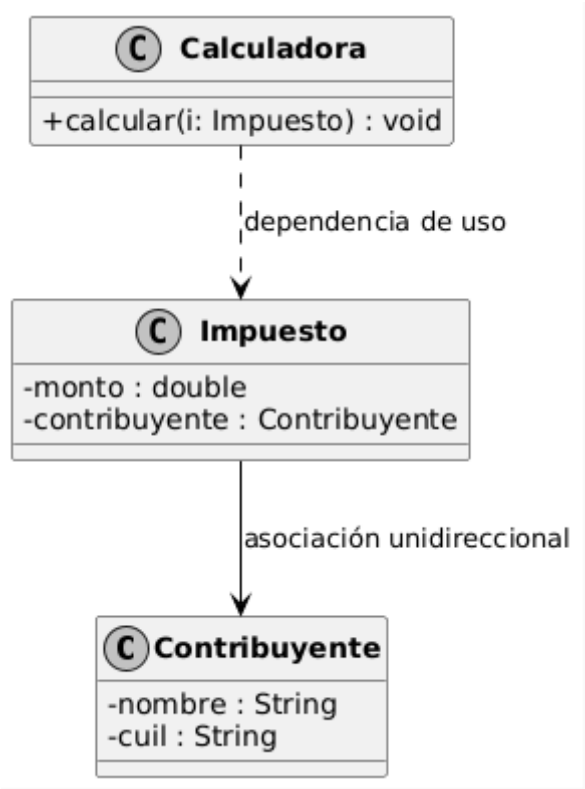
1. **Asociación unidireccional:**
   Impuesto → Contribuyente
   - Impuesto *conoce* al Contribuyente.

- Contribuyente **no** conoce al Impuesto.

2. **Dependencia de uso:**
   Calculadora.calcular(Impuesto)
   - Calculadora *usa* a Impuesto como parámetro, pero **no lo almacena como atributo**.

```
// ----------------------
// Clase Contribuyente
// ----------------------
public class Contribuyente {
    private String nombre;
    private String cuil;

    public Contribuyente(String nombre, String cuil) {
        this.nombre = nombre;
        this.cuil = cuil;
    }

    public String getNombre() {
        return nombre;
    }

    public String getCuil() {
        return cuil;
    }

    @Override
    public String toString() {
        return nombre + " (CUIL: " + cuil + ")";
    }
}
```

Diagrama UML



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)

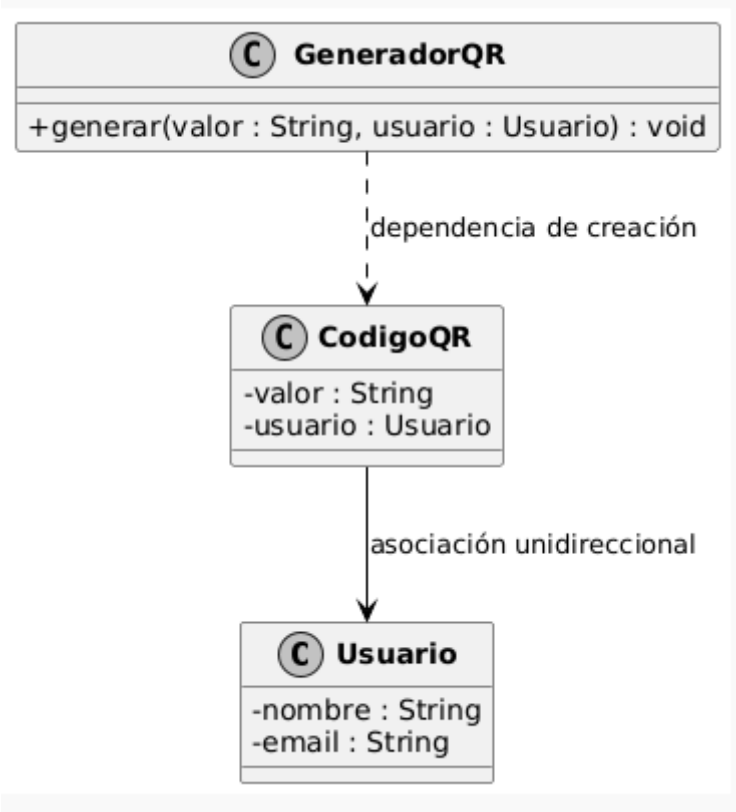
Relaciones pedidas

1. **Asociación unidireccional:**
   CodigoQR → Usuario
   → La clase CodigoQR *conoce* al Usuario que lo generó, pero el Usuario **no conoce** al CodigoQR.
2. **Dependencia de creación:**
   GeneradorQR.generar(String valor, Usuario usuario)
   → GeneradorQR *crea* un objeto CodigoQR dentro de su método generar.
   (No lo guarda como atributo; solo lo instancia y lo usa temporalmente.)

```
// ----------------------
// Clase Usuario
// ----------------------
public class Usuario {
    private String nombre;
    private String email;

    public Usuario(String nombre, String email) {
        this.nombre = nombre;
        this.email = email;
    }

    public String getNombre() {
        return nombre;
    }

    public String getEmail() {
        return email;
    }

    @Override
    public String toString() {
```

```
        return nombre + " (" + email + ")";
    }
}
```

Diagrama UML



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)

Relaciones

1. **Asociación unidireccional:**
   Render → Proyecto
   → La clase Render *conoce* al Proyecto que se está exportando, pero Proyecto **no conoce** a Render.

2. **Dependencia de creación:**
   EditorVideo.exportar(String formato, Proyecto proyecto)
   → La clase EditorVideo **crea** un objeto Render dentro de su método exportar.
   (No lo guarda como atributo, solo lo instancia temporalmente.)

```
// ----------------------
// Clase Proyecto
// ----------------------
public class Proyecto {
    private String nombre;
    private int duracionMin; // duración en minutos

    public Proyecto(String nombre, int duracionMin) {
        this.nombre = nombre;
        this.duracionMin = duracionMin;
    }

    public String getNombre() {
        return nombre;
    }

    public int getDuracionMin() {
        return duracionMin;
    }

    @Override
    public String toString() {
        return nombre + " (" + duracionMin + " min)";
    }
```

```
}

// ----------------------
// Clase Render
// ----------------------
public class Render {
    private String formato;
    private Proyecto proyecto; // Asociación unidireccional

    public Render(String formato, Proyecto proyecto) {
        this.formato = formato;
        this.proyecto = proyecto;
    }

    public String getFormato() {
        return formato;
    }

    public Proyecto getProyecto() {
        return proyecto;
    }

    @Override
    public String toString() {
        return "Render en formato [" + formato + "] del proyecto " +
proyecto.getNombre();
    }
}
```

Diagrama UML