Apunte – Relaciones 1 a 1, Dependencia y "Tell, Don't Ask" en Java y UML

1. Relaciones entre Clases en POO

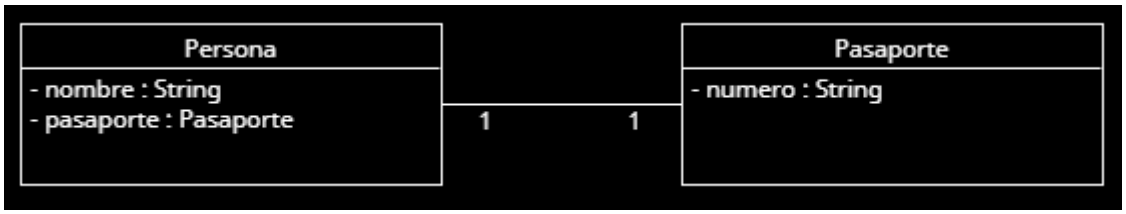
En Programación Orientada a Objetos, las clases no existen de forma aislada: interactúan entre sí. Una manera de expresar esa interacción es a través de **atributos que son objetos de otras clases**.

Ejemplo:
Un **Pasaporte** siempre pertenece a una **Persona**.

En Java:

```java
public class Persona {

    private String nombre;
    private Pasaporte pasaporte; // atributo de tipo objeto
}

public class Pasaporte {

    private String numero;
}
```

En UML (relación **1 a 1**):



2. Relaciones 1 a 1 en UML

Una **relación 1 a 1** significa que **una instancia de una clase está asociada con una única instancia de otra clase**.

Existen distintos tipos:

Asociación unidireccional

Solo una clase conoce a la otra.
Ejemplo: Una **Persona** conoce su **Pasaporte**, pero el **Pasaporte** no conoce a la **Persona**.

Asociación bidireccional

Ambas clases se conocen.
Ejemplo: Una **Persona** conoce su **Pasaporte**, y el **Pasaporte** conoce a la **Persona**.

Agregación

Una clase forma parte de otra, pero puede existir por separado.
Ejemplo: Un **Departamento** pertenece a una **Universidad**, pero puede existir sin ella.

Composición

Una clase depende totalmente de otra: si se elimina una, la otra también desaparece.
Ejemplo: Un **Corazón** no puede existir sin una **Persona**.

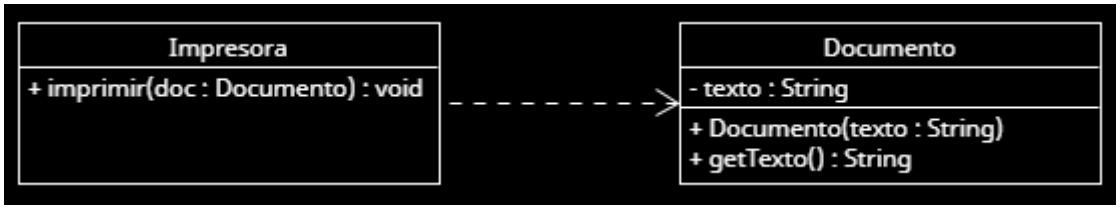
Dependencia en Java

La **dependencia** indica que una clase **usa** otra clase en un momento determinado, sin necesidad de tenerla como atributo permanente.

Ejemplo:

```java
public class Impresora {

    public void imprimir(Documento doc) {
        System.out.println("Imprimiendo: " + doc.getTexto());
    }
}

public class Documento {

    private String texto;

    public Documento(String texto) {
        this.texto = texto;
    }

    public String getTexto() {
        return texto;
    }
}
```

La clase Impresora depende de Documento solo para ejecutar el método imprimir.
En UML, la dependencia se representa con una **flecha discontinua**.



4. Principio "Tell, Don't Ask"

Este principio propone:

- **No pedir datos a un objeto para procesarlos afuera.**
- En cambio, **decirle al objeto qué hacer**.

Ejemplo de mal diseño

```
// Mal: pedimos información para procesarla en otro lugar
double saldo = cuenta.getSaldo();
if (saldo >= 100) {
    cuenta.setSaldo(saldo - 100);
}
```

Ejemplo de buen diseño

```
// Bien: le decimos a la cuenta lo que tiene que hacer
cuenta.retirar(100);
```

Ventajas:

- Se mejora el **encapsulamiento**.
- Se reduce el **acoplamiento** entre clases.
- El código se vuelve más **claro y mantenible**.