

# Apunte Relaciones 1 a 1 en Java

*Guía completa con ejemplos de codificación, niveles de acoplamiento y buenas prácticas.*

## ◆ 1. Relaciones Unidireccionales

En las relaciones unidireccionales, **solo una de las clases conoce a la otra**. Se representan mediante una **referencia simple** y no existe retroalimentación entre objetos.

Las relaciones unidireccionales se clasifican según su nivel de acoplamiento:

Tipo de relación	Fortalezas del vínculo	Nivel de acoplamiento
Asociación	Débil	Bajo
Agregación	Medio	Medio
Composición	Fuerte	Alto

### ☒ Asociación Unidireccional

✦ **Definición:** Una clase contiene una referencia a otra, pero la otra no tiene conocimiento de esta.

✦ **Implementación:** Se establece mediante un `setter` desde el `main`.

**Ejemplo: Alumno y Tutor**

```
public class Tutor {  
    private String nombre;  
  
    public Tutor(String nombre) {  
        this.nombre = nombre;  
    }  
  
    public String getNombre() {  
        return nombre;  
    }  
}  
  
public class Alumno {  
    private String nombre;  
    private Tutor tutor;  
  
    public Alumno(String nombre) {  
        this.nombre = nombre;  
    }  
}
```

```
    }

    public void setTutor(Tutor tutor) {
        this.tutor = tutor;
    }

    public void mostrar() {
        System.out.println("Alumno: " + nombre + " - Tutor: " +
tutor.getNombre());
    }
}
public class Main {
    public static void main(String[] args) {
        Tutor tutor = new Tutor("Prof. Gómez");
        Alumno alumno = new Alumno("Lucía");
        alumno.setTutor(tutor);
        alumno.mostrar();
    }
}
```

---

## ☒ Agregación

✦ **Definición:** Una clase está compuesta por otra, pero esta puede existir por separado. Hay una **relación de pertenencia flexible**.

✦ **Implementación:** Se pasa la instancia como **parámetro en el constructor**.

### Ejemplo: Curso y Profesor

```
public class Profesor {
    private String nombre;

    public Profesor(String nombre) {
        this.nombre = nombre;
    }

    public String getNombre() {
        return nombre;
    }
}

public class Curso {
    private String nombre;
    private Profesor profesor;


    public Curso(String nombre, Profesor profesor) {
        this.nombre = nombre;
        this.profesor = profesor;
    }

    public void mostrar() {
        System.out.println("Curso: " + nombre + " - Profesor: " +
profesor.getNombre());
    }
}
```

```
}  
  
public class Main {  
    public static void main(String[] args) {  
        Profesor profe = new Profesor("Fernández");  
        Curso curso = new Curso("Matemática", profe);  
        curso.mostrar();  
    }  
}
```

---

## Composición

 **Definición:** Una clase **posee y controla el ciclo de vida** de la otra. Si el "todo" se destruye, las partes también.

 **Implementación:** Se **crea internamente** la instancia a través de parámetros primitivos del constructor.

### Ejemplo: Pasaporte y Foto

```
public class Foto {  
    private String url;  
  
    public Foto(String url) {  
        this.url = url;  
    }  
  
    public String getUrl() {  
        return url;  
    }  
}  
  
public class Pasaporte {  
    private String numero;  
    private Foto foto;  
  
    public Pasaporte(String numero, String urlFoto) {  
        this.numero = numero;  
        this.foto = new Foto(urlFoto);  
    }  
  
    public void mostrar() {  
        System.out.println("Pasaporte: " + numero + " - Foto: " +  
foto.getUrl());  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {  
        Pasaporte pasaporte = new Pasaporte("A123456", "foto.jpg");  
        pasaporte.mostrar();  
    }  
}
```

## 2. Relaciones 1 a 1 Bidireccional

✦ **Definición:** Ambas clases tienen una referencia mutua. Se requiere **sincronización de datos** para mantener coherencia.

✦ **Nivel de acoplamiento:** Medio a alto.

✦ **Buena práctica:** Usar setters con validación para mantener la coherencia de la relación.

### ☒ **Ejemplo: Pasaporte y Titular**

```
public class Titular {
    private String nombre;
    private Pasaporte pasaporte;

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

    public void setPasaporte(Pasaporte pasaporte) {
        this.pasaporte = pasaporte;
        if (pasaporte != null && pasaporte.getTitular() != this) {
            pasaporte.setTitular(this);
        }
    }

    public String getNombre() {
        return nombre;
    }

    public Pasaporte getPasaporte() {
        return pasaporte;
    }
}

public class Pasaporte {
    private String numero;
    private Titular titular;

    public Pasaporte(String numero) {
        this.numero = numero;
    }

    public void setTitular(Titular titular) {
        this.titular = titular;
        if (titular != null && titular.getPasaporte() != this) {
            titular.setPasaporte(this);
        }
    }
}
```

```

    }

    public String getNumero() {
        return numero;
    }

    public Titular getTitular() {
        return titular;
    }
}

public class Main {
    public static void main(String[] args) {
        Titular titular = new Titular("Ana López");
        Pasaporte pasaporte = new Pasaporte("X123456");
        titular.setPasaporte(pasaporte); // Solo un setter necesario
        System.out.println("Titular: " + titular.getNombre());
        System.out.println("Pasaporte: " +
titular.getPasaporte().getNumero());
    }
}

```



### 3. Comparación de relaciones 1 a 1

Tipo de relación	Dirección	Fortaleza del vínculo	Ciclo de vida compartido	Implementación recomendada
Asociación	Unidireccional	Débil	No	Setter en el main
Agregación	Unidireccional	Medio	No	Constructor con objeto externo
Composición	Unidireccional	Fuerte	Sí	Constructor con datos primitivos
Asociación Bidireccional	Bidireccional	Medio	Parcialmente compartido	Setters sincronizados



### 4. Dependencias en UML

Las dependencias representan **usos temporales** entre clases. No implican relación permanente.



#### Dependencia de uso

```

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

```

```
public class Documento {  
    private String contenido;  
    public Documento(String contenido) {  
        this.contenido = contenido;  
    }  
    public String getContenido() {  
        return contenido;  
    }  
}
```

---

### Dependencia de creación

```
public class GestorUsuario {  
    public void crearUsuario(String nombre) {  
        Usuario u = new Usuario(nombre);  
        System.out.println("Usuario: " + u.getNombre());  
    }  
}  
public class Usuario {  
    private String nombre;  
    public Usuario(String nombre) {  
        this.nombre = nombre;  
    }  
    public String getNombre() {  
        return nombre;  
    }  
}
```

---

### Comparación

Tipo de dependencia	Uso	Acoplamiento	Persistencia de la relación
Dependencia de uso	Uso temporal	Bajo	No
Dependencia de creación	Creación interna	Medio	No