



Apunte: Relaciones 1 a N en Java

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

1. Relaciones Unidireccionales 1 a N

En una relación 1 a N, una instancia de una clase se vincula con varias instancias de otra clase. Las relaciones se clasifican según su nivel de acoplamiento y la fortaleza del vínculo.

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

✓ Asociación Unidireccional 1 a N

→ Definición: Una clase conoce a varias instancias de otra clase, pero no existe un vínculo de dependencia fuerte.

★ Implementación: Se utiliza una colección (ej. List) y se agregan los objetos mediante métodos add desde el main.

Ejemplo: Profesor y Curso

```
public class Curso {
    private String nombre;
    private int creditos;

public Curso(String nombre, int creditos) {
        this.nombre = nombre;
        this.creditos = creditos;
    }

    public String getNombre() { return nombre; }
    public int getCreditos() { return creditos; }
}
```



```
public class Profesor {
   private String nombre;
   private List<Curso> cursos = new ArrayList<>();
   public Profesor(String nombre) {
       this.nombre = nombre;
   public void agregarCurso(Curso curso) {
       cursos.add(curso);
   public void mostrarCursos() {
       System.out.println("Los cursos del profesor "
       +nombre+" son: ");
       for(Curso curso: getCursos()){
          System.out.println("Nombre del curso: "+
                 curso.getNombre());
          System.out.println("Creditos del curso: "+
                 curso.getCreditos());
          System.out.println("----");
       }
  }
public class Main {
public static void main(String[] args) {
       //Instanciamos un Profesor
       Profesor marcos=new Profesor("Marcos Marconi");
       //Instanciamos varios cursos
      Curso programacion1= new Curso ("Programacion 1", 20);
      Curso ayso= new Curso ("Arquitectura y Sistemas
Operativos", 10);
      Curso programacion2= new Curso("Programacion 2",30);
       //Asociamos los cursos al Profesor
      marcos.agregarCurso(programacion1);
      marcos.agregarCurso(programacion2);
      marcos.agregarCurso(ayso);
      marcos.mostrarCursos();
      marcos.eliminarCurso(ayso);
       System.out.println("Imprimimos los cursos despues de eliminar
A&SO");
       marcos.mostrarCursos();
```



✓ Agregación 1 a N

→ Definición: Una clase contiene varias instancias de otra clase, pero estas pueden existir por separado.

★ Implementación: Se gestiona la colección en el constructor o con métodos add, validando nulls y duplicados.

Ejemplo: Empresa y Empleados

```
public class Empleado {
    private String nombre;
   private String puesto;
    public Empleado(String nombre, String puesto) {
        this.nombre = nombre;
        this.puesto = puesto;
    public String getNombre() { return nombre; }
    public String getPuesto() { return puesto; }
public class Empresa {
    private String nombre;
    private List<Empleado> empleados = new ArrayList<>();
    public Empresa(String nombre) {
        this.nombre = nombre;
    public void agregarEmpleado(Empleado emp) {
        if (emp != null && !empleados.contains(emp)) {
            empleados.add(emp);
    public void removerEmpleado(Empleado emp) {
            if(emp!=null && empleados.contains(emp)){
                empleados.remove(emp);
        }
    public void mostrarEmpleado(){
        System.out.println("La empres "+ nombre
```



```
+" tiene los siguientes empleados:");
       System.out.println("|||||||||||||||||||);
       for(Empleado emp: empleados){
          System.out.println("Empleado: "+emp.getNombre());
          System.out.println("Puesto: "+emp.getPuesto());
          System.out.println("----");
       }
   }
}
public class main{
public static void main(String[] args) {
       //Instanciamos la Empresa
      Empresa claro= new Empresa("Claro");
      //Instanciamos empleados
      Empleado carlos= new Empleado ("Carlos Casares", "Tecnico
Instalador");
      Empleado maria = new Empleado ("Maria Perez", "Agente de Ventas");
      Empleado juana= new Empleado ("Juana Viale", "Gerente de
Desarrollo de Red");
      //Agregamos los empleados a la Empresa
      claro.agregarEmpleado(juana);
      claro.agregarEmpleado(carlos);
      claro.agregarEmpleado(maria);
      //Mostramos por pantalla la Empresa
      claro.mostrarEmpleado();
       System.out.println("///////////////;);
       System.out.println("Eliminamos un empleado");
       claro.removerEmpleado(carlos);
       claro.mostrarEmpleado();
       System.out.println("Imprimimos el empleado eliminado para ver
que existe en mi programa");
       System.out.println("Empleado despedido: "+carlos.getNombre());
}
```

☑ Composición 1 a N

→ Definición: Una clase controla completamente la creación y destrucción de las instancias contenidas.

★ Implementación: Se crean internamente los objetos dentro de la clase contenedora a partir de datos primitivos.

Ejemplo: Pedido e Ítems

```
public class Item {
    private String nombre;
    private int cantidad;
```



```
private double precio;
    public Item(String nombre, int cantidad, double precio) {
        this.nombre = nombre;
        this.cantidad = cantidad;
        this.precio = precio;
    public double getSubtotal() {
       return cantidad * precio;
    public String getNombre() { return nombre; }
}
public class Pedido {
   private String codigo;
   private List<Item> items = new ArrayList<>();
   public Pedido(String codigo) {
        this.codigo = codigo;
   public void agregarItem(String nombre, int cantidad, double
precio) {
        items.add(new Item(nombre, cantidad, precio));
public boolean removeItemPorNombre(String nombreItem) {
        for (Item item : items) {
            if (item.getNombre().equals(nombreItem)) {
                items.remove(item);
                return true;
        return false;
    public void mostrarTotal() {
        double total = 0;
        for (Item i : items) {
           total += i.getSubtotal();
        System.out.println("Total del pedido: $" + total);
public void mostrarPedido(){
        System.out.println("El pedido: "+codigo
        +" con fecha "+fecha+" Precio Total: "+calcularTotal());
        for(Item item:items) {
            System.out.println("Item: "+item.getNombre()
            +" precio: "+item.getPrecio());
            System.out.println("cantidad: "+item.getCantidad()+"
subtotal: "+item.getSubtotal());
            System.out.println("----");
    }
```



```
}
public class main{
public static void main(String[] args) {
        //Instanciamos un Pedido
       Pedido pedidoPc= new Pedido("PC500", new Date());
       //Agregamos Items al pedido por composicion
       pedidoPc.agregarItem("Placa Madre", 1, 300);
       pedidoPc.agregarItem("Memoria Ram", 2, 400);
       pedidoPc.agregarItem("Procesador", 1, 400);
       //Mostramos el pedido
       pedidoPc.mostrarPedido();
       //Removemos un Item
       //No hay referencia externa por lo que al borrarse no existe
mas en mi programa
       pedidoPc.removeItemPorNombre("Placa Madre");
       //Mostramos el pedido despues de borrar un item
        System.out.println("El pedido despues de borrar un Item");
       pedidoPc.mostrarPedido();
    }
}
```

2. Relaciones Bidireccionales 1 a N

▶ Definición: Ambas clases se conocen mutuamente. Se requiere sincronizar ambas direcciones para mantener la coherencia.

Ejemplo: Departamento y Empleado

```
public class Empleado {
    private String nombre;
    private Departamento departamento;

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

public void setDepartamento(Departamento departamento) {
        if (this.departamento == departamento) return;

        if (this.departamento != null) {
            this.departamento.eliminarEmpleado(this);
        }

        this.departamento = departamento;

        if (departamento != null &&
!departamento.getEmpleados().contains(this)) {
```



```
departamento.agregarEmpleado(this);
        }
    }
    public Departamento getDepartamento() {
       return departamento;
   public String getNombre() {
       return nombre;
}
public class Departamento {
   private String nombre;
   private List<Empleado> empleados = new ArrayList<>();
   public Departamento(String nombre) {
        this.nombre = nombre;
   public void agregarEmpleado(Empleado emp) {
        if (emp != null && !empleados.contains(emp)) {
           empleados.add(emp);
           if (emp.getDepartamento() != this) {
               emp.setDepartamento(this);
        }
    public void eliminarEmpleado(Empleado emp) {
        if (empleados.remove(emp) && emp.getDepartamento() == this) {
           emp.setDepartamento(null);
    }
   public List<Empleado> getEmpleados() {
       return Collections.unmodifiableList(empleados);
    public String getNombre() {
       return nombre;
public void mostrarDepartamento(){
       System.out.println("El departamento "+nombre+" tiene los
siguientes empleados");
        for(Empleado emp: empleados){
           System.out.println("Empleado "+emp.getNombre());
            System.out.println("Puesto "+emp.getPuesto());
            //Validamos que el empleado tambien asocio el departamento
           System.out.println("Departamento
"+emp.getDepartamento().getNombre());
           System.out.println("----");
        }
   }
}
```



```
public class main{
public static void main(String[] args) {
       //Instanciamos 2 Departamento
       Departamento ventas= new Departamento("Ventas");
       Departamento rrhh= new Departamento("RR.HH");
       //Instanciamos empleados
       Empleado mario= new Empleado("Mario Baracus", "Vendedor");
       Empleado maria= new Empleado("Maria Fara", "Reclutador");
       Empleado carlos= new Empleado ("Carlos Castro", "Gerente de
Ventas");
       Empleado rocio= new Empleado ("Rocio Mercado", "Gerente RRHH");
       //Asociamos los empleados a los Departamentos
       ventas.agregarEmpleado(carlos);
       ventas.agregarEmpleado(mario);
       rrhh.agregarEmpleado (maria);
       rrhh.agregarEmpleado(rocio);
       //Mostramos los departamentos
       ventas.mostrarDepartamento();
       System.out.println("|||||||||||||);
       rrhh.mostrarDepartamento();
       //Traspasamos un empleado de Ventas a RRHH
       carlos.setDepartamento(rrhh);
       //Mostramos de nuevo los departamentos
       ventas.mostrarDepartamento();
       System.out.println("||||||||||||);
       rrhh.mostrarDepartamento();
```

3. 🙀 Comparación de Relaciones 1 a N

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	Método add con validación
Composición	Unidireccional	Fuerte	X 1	Constructor con datos primitivos
Bidireccional	Bidireccional	Medio	Parcialmente	Setters sincronizados