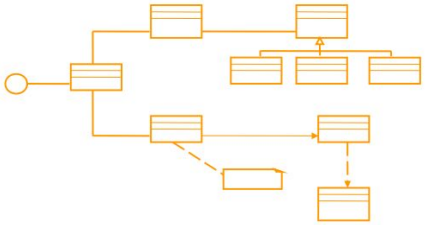


Relaciones entre Clases

UNJu - Programación Orientada a Objetos

Ing. José Zapana

Visibilidad de atributos y operaciones de una clase

- La visibilidad puede ser:
  - Público (+): objetos de cualquier clase.
  - Protegido (#): objetos de las subclases.
  - De paquete (~): objetos de cualquier clase del mismo paquete.
  - Privado (-): objetos donde está definido el atributo u operación.

MODIFICADOR	CLASE	PACKAGE	SUBCLASE	TODOS
public	Sí	Sí	Sí	Sí
protected	Sí	Sí	Sí	No
No especificado	Sí	Sí	No	No
private	Sí	No	No	No

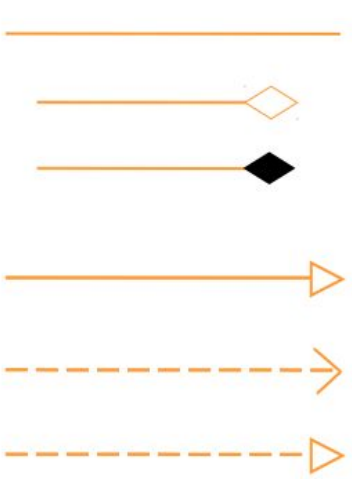
Introducción

- En la POO las clases no existen en forma aislada: suelen interactuar entre sí y estas interacciones se representan como relaciones.
- Las relaciones son una conexión semántica entre objetos. Proveen un camino de comunicación entre ellos.
- Comprenderlas permite diseñar software más modular, escalable y mantenible.

Tipos de relaciones entre clases

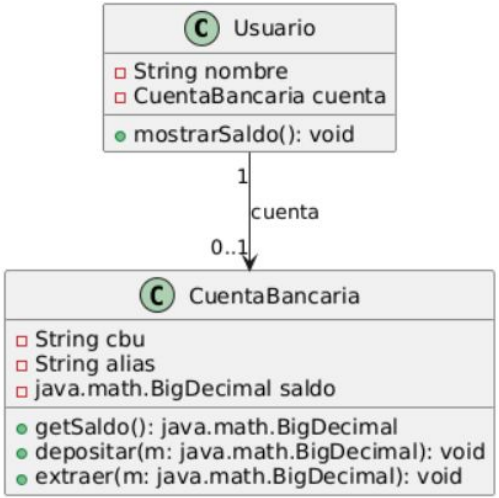
Notación

- Asociación
  - Agregación
  - Composición
- Generalización
- Dependencia
- Realización



Asociación

- Una clase se relaciona con otra para cumplir una función.
- Ejemplo: Un **Usuario** puede tener una **CuentaBancaria**.




```java
class Usuario {
    private String nombre;
    private CuentaBancaria cuenta;

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

    public void mostrarSaldo() {
        System.out.println("Saldo: " + cuenta.getSaldo());
    }
}
```

```java
class CuentaBancaria {
    private double saldo;
    public CuentaBancaria(double saldo) { this.saldo = saldo; }
    public double getSaldo() { return saldo; }
}
```

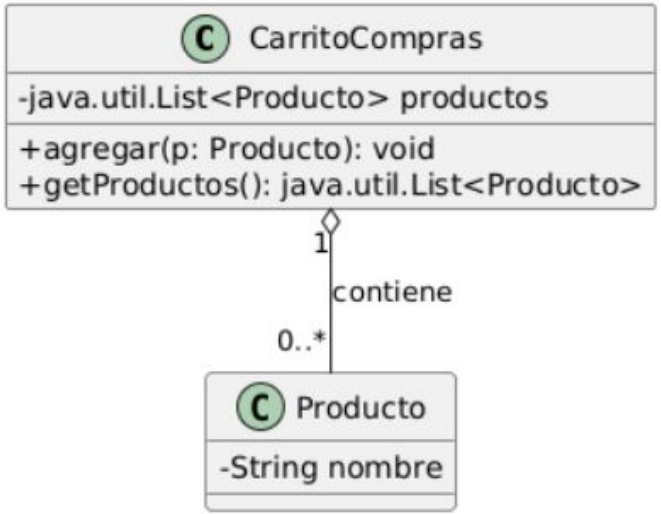
Agregación

- Relación “tiene un” pero débil, los objetos pueden existir independientemente.
- Ejemplo: Un **CarritoDeCompras** contiene **Productos**, pero los productos existen sin el carrito.

```
class CarritoDeCompras {
    private List<Producto> productos = new ArrayList<>();

    public void agregarProducto(Producto p) {
        productos.add(p);
    }
}

class Producto {
    private String nombre;
    public Producto(String nombre) { this.nombre = nombre; }
}
```



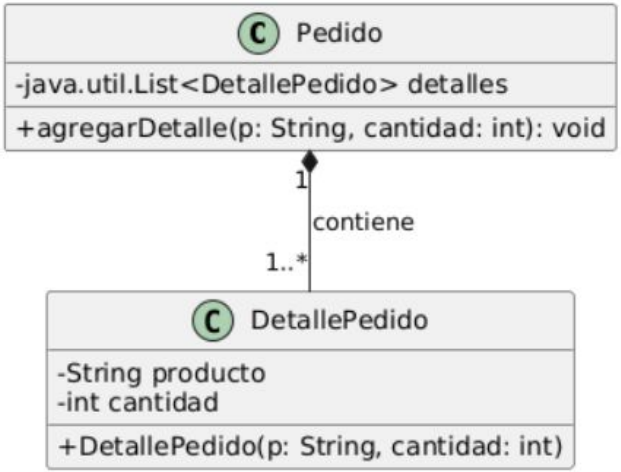
Composición

- Relación fuerte, el ciclo de vida de un objeto depende del otro.
- Ejemplo: Un **Pedido** tiene **DetallePedidos** que no existen fuera del pedido..

```java
class Pedido {
    private List<DetallePedido> detalles = new ArrayList<>();

    public void agregarDetalle(String producto, int cantidad) {
        detalles.add(new DetallePedido(producto, cantidad));
    }
}
```

```java
class DetallePedido {
    private String producto;
    private int cantidad;
    public DetallePedido(String producto, int cantidad) {
        this.producto = producto;
        this.cantidad = cantidad;
    }
}
```

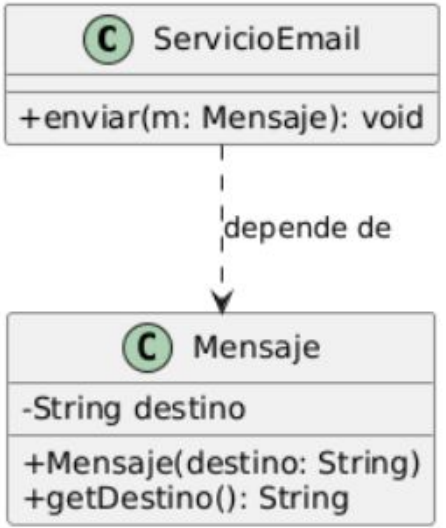


Dependencia

- Una clase usa otra de manera temporal, solo como parámetro o dentro de un método.
- Ejemplo: Un **ServicioEmail** depende de **Mensaje** para enviarlo, pero no lo almacena permanentemente.

```
class ServicioEmail {
    public void enviar(Mensaje mensaje) {
        System.out.println("Enviando email a " + mensaje.getDestino());
    }
}

class Mensaje {
    private String destino;
    public Mensaje(String destino) { this.destino = destino; }
    public String getDestino() { return destino; }
}
```



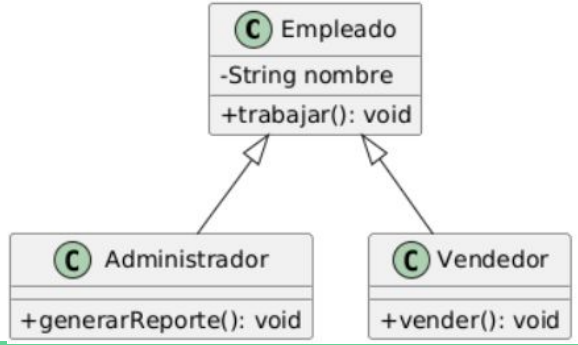
Herencia

- Define una relación de clases, en la que una clase comparte los **comportamientos** y la **estructura** de datos de la otra
- Constituye una técnica valiosa, porque posibilita y fomenta la reutilización del software

```java
class Empleado {
    protected String nombre;
    public void trabajar() {
        System.out.println(nombre + " está trabajando.");
    }
}
```

```java
class Vendedor extends Empleado {
    public void vender() {
        System.out.println("Realizando una venta...");
    }
}
```

```java
class Administrador extends Empleado {
    public void generarReporte() {
        System.out.println("Generando reporte...");
    }
}
```

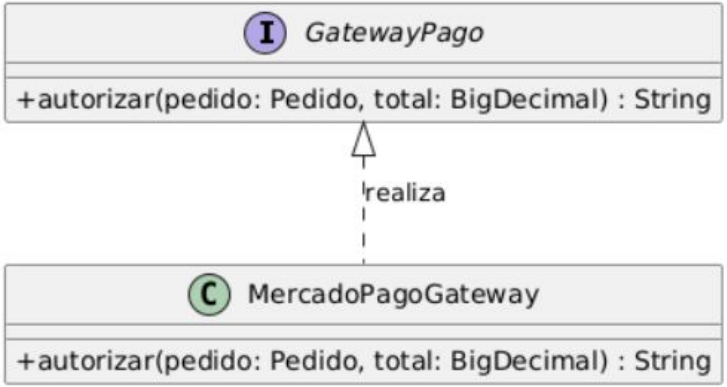


Realización

- Es una relación donde un clasificador concreto cumple el contrato definido por otro clasificador más abstracto.
- En diagramas de clases, la forma típica es Clase que implementa una Interfaz

```java
public interface GatewayPago {
    String autorizar(Pedido pedido, BigDecimal total);
}
```

```java
public class MercadoPagoGateway implements GatewayPago {
    @Override
    public String autorizar(Pedido pedido, BigDecimal total) {
        // mock: en un caso real, llamás al SDK/REST de MercadoPago
        return "MP-" + pedido.getNumero();
    }
}
```



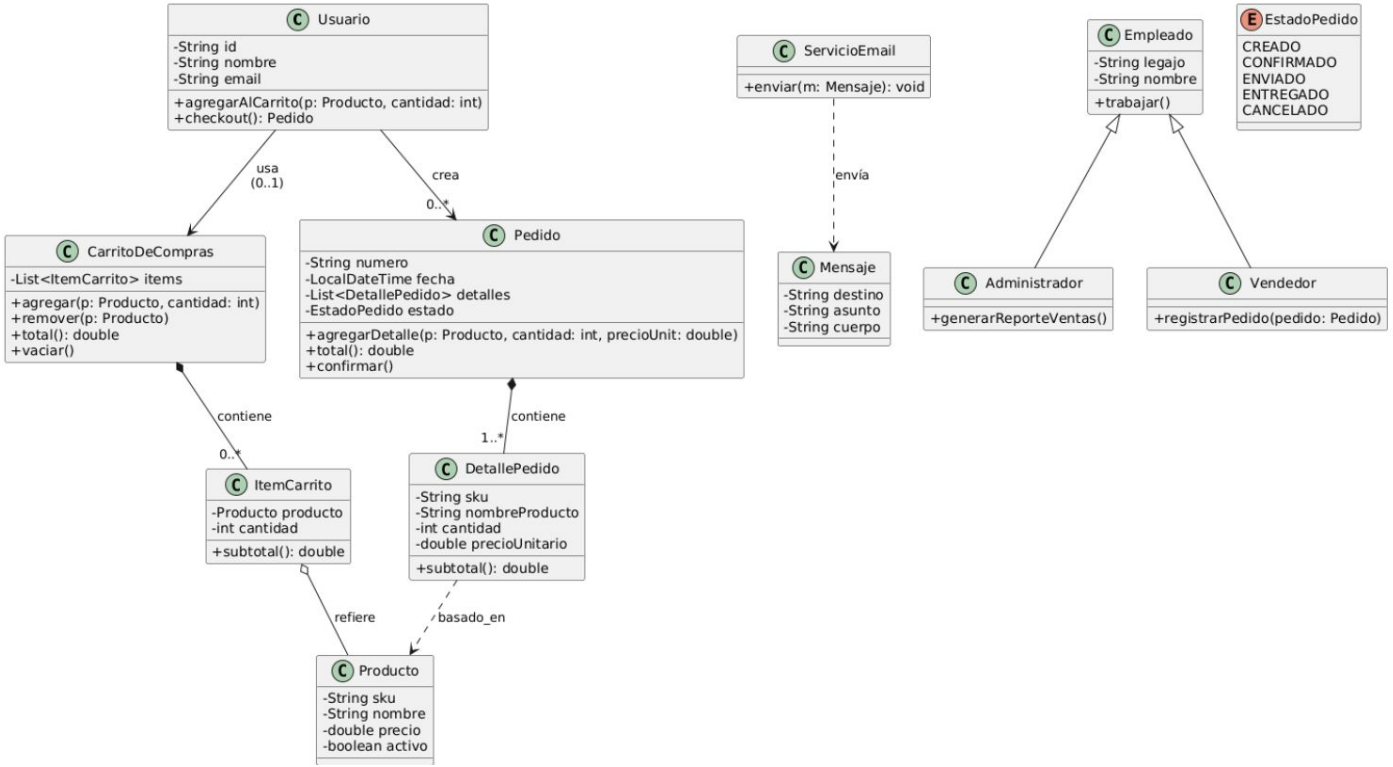
Resumen comparativo

Relación	Ejemplo Software	Nivel de acoplamiento	Ciclo de vida
Asociación	`Usuario` ↔ `CuentaBancaria`	Medio	Independiente
Agregación	`CarritoDeCompras` → `Producto`	Medio-Bajo	Independiente
Composición	`Pedido` → `DetallePedido`	Alto	Dependiente
Herencia	`Empleado` → `Vendedor`	Alto	Hijo hereda
Dependencia	`ServicioEmail` → `Mensaje`	Bajo	Temporal
Realización	`MercadoPagoGateway` **realiza** `GatewayPago` ; `ServicioPago` usa `GatewayPago`	**Bajo** (se programa contra interfaz)	Independiente

Caso Integrado: E-commerce

- Clases: Usuario, CarritoDeCompras, Producto, Pedido, DetallePedido, ServicioEmail.
- Relaciones:
  - **Usuario** tiene un **CarritoDeCompras** (Asociación).
  - **CarritoDeCompras** contiene **Productos** (Agregación).
  - **Pedido** contiene **DetallePedido** (Composición).
  - **Empleados** especializados (Herencia).
  - **ServicioEmail** envía **Mensajes** (Dependencia).

Caso Integrado: E-commerce Diagrama UML

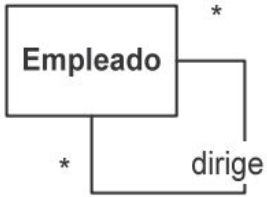


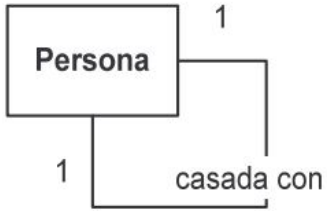
Consideraciones Importantes

- La herencia sólo se debe utilizar para relaciones “es un tipo de” reales:
  - Debe ser siempre posible sustituir un objeto de subclase por uno de superclase.
  - Todos los métodos de la superclase deben tener sentido en la subclase
- El uso de la herencia como solución a corto plazo provoca problemas en el futuro.

Autoasociación (reflexiva)

- Es una asociación donde ambos extremos son la misma clase
- Sirve para modelar jerarquías del mismo tipo (categorías padre/hija), estructuras de organización (manager ⇄ subordinados), grafos (nodo conectado a nodo), etc.






```java
public class Empleado {
    private Integer id;
    private String nombre;
    private Empleado responsable;
    public Integer getId() {
        return id;
    }
```

id	nombre	id_responsable
1	A	
2	B	1
3	C	1
4	D	2
5	E	2

Cado de estudio Herencia