



# Algoritmos y Estructura de Datos

---

Unidad 4: Técnicas Avanzadas de POO

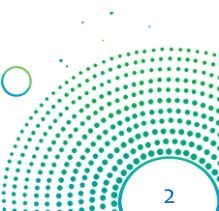
Tema 12: Herencia y polimorfismo





# Tema 12: Herencia y polimorfismo

---





# Índice

---

## 4.1 Tema 12: Herencia y polimorfismo

- 4.1.1. Generalización / Especialización
- 4.1.2. Herencia
- 4.1.3. Relación “es un” o “es una”
- 4.1.4. Uso de *super*  
Ejemplo\_A
- 4.1.5. Uso de *super* en sobre escritura
- 4.1.6. Operador *instanceof*
- 4.1.7. Polimorfismo  
Ejemplo\_B





# Capacidades

---

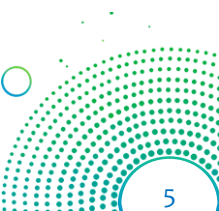
- Implementa herencia simple utilizando superclases concretas.
- Aplica polimorfismo básico.





# 4.1.1 Generalización / Especialización

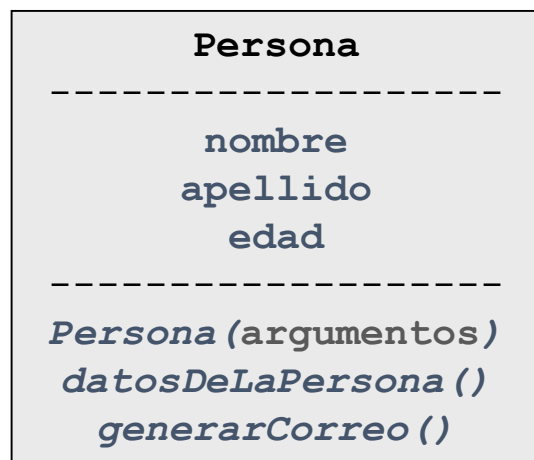
- La relación Generalización / Especialización se produce cuando dos o más clases tienen muchas de sus partes en común (atributos y métodos) lo que normalmente se deriva en la creación de una nueva clase que reúne todas sus características comunes.
- La relación Generalización / Especialización es la relación de una clase más general y una clase más específica. La clase más general se denomina **superClase** o clase **Padre** y la clase más específica se denomina **subClase** o Clase **Hijo** quien además posee información adicional.
- La Generalización / Especialización son diferentes vistas de un mismo concepto. La Generalización es una perspectiva ascendente (*bottom-up*), mientras que la Especialización es una perspectiva descendente (*top-down*).



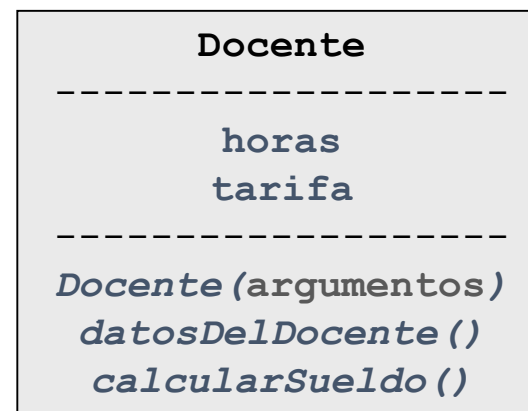
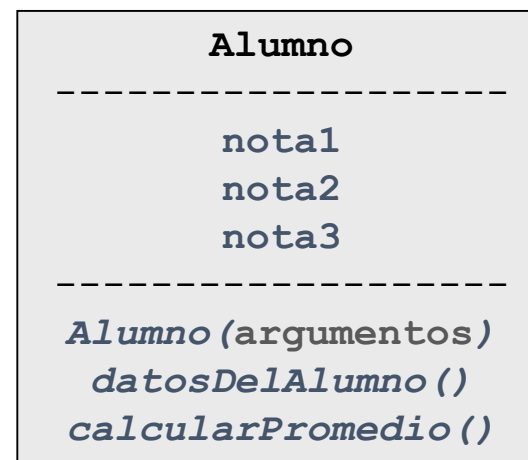


# 4.1.1 Generalización / Especialización

## Generalización



## Especialización



- **Persona** es una superClase (Padre)
- **Alumno** es una subClase (Hijo)
- **Docente** es una subClase (Hijo)



## 4.1.2 Herencia

- Es el mecanismo mediante el cual se puede definir una clase (**Hijo**) sobre la base de otra clase (**Padre**), heredando aquellos miembros de la clase **Padre** (atributos y métodos) que hayan sido declarados como *public*, *protected* o sin especificador de acceso.
- Una clase **Padre** declara un miembro como *protected* para permitir el acceso al miembro desde el interior de la clase **Hijo** y desde una clase que se encuentra en el mismo paquete, a la vez que impide el acceso al miembro desde el exterior del paquete.
- Las clases **Hijo** heredan características de las clases **Padre** y añaden características específicas que las diferencian.
- Las clases se organizan en una estructura jerárquica.
- El constructor no es un miembro, por tanto las clases **Hijo** no lo pueden heredar.






## 4.1.2 Herencia

- La forma general de la declaración de una clase **Hijo** que hereda de una clase **Padre** es la siguiente:

```
public class Padre {  
    // Cuerpo de la superClase  
    ...  
}
```

```
public class Hijo extends Padre {  
    // Cuerpo de la subClase  
    ...  
}
```



```
public class Persona {  
    // Cuerpo de la superClase  
    ...  
}  
public class Alumno extends Persona {  
    // Cuerpo de la subClase  
    ...  
}
```








## 4.1.2 Herencia

- Si una clase no tiene una clase **Padre** explícita, entonces implícitamente su clase **Padre** es la clase *Object*.
- La clase *Object* define e implementa un comportamiento requerido por todas las clases dentro del Sistema Java.
- Así, en el caso de la clase *Persona*, implícitamente figura como:

```
public class Padre extends Object {  
    // Cuerpo de la superClase  
    ...  
}
```

```
public class Hijo extends Padre {  
    // Cuerpo de la subClase  
    ...  
}
```



```
public class Persona extends Object {  
    // Cuerpo de la superClase  
    ...  
}  
public class Alumno extends Persona {  
    // Cuerpo de la subClase  
    ...  
}
```





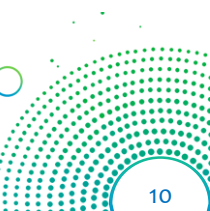
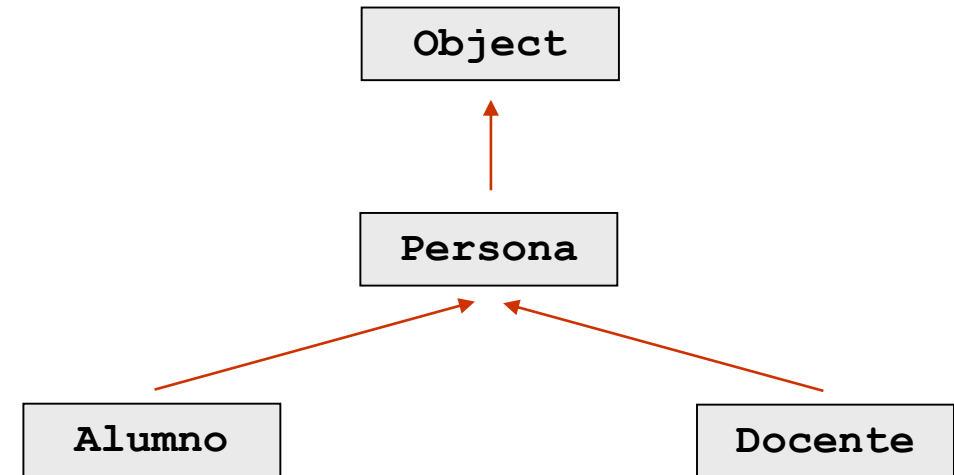
## 4.1.3 Relación “es un” o “es una”

- La herencia permite establecer una jerarquía de especialización mediante la relación “*es-un*” o “*es-una*”.

### Caso 1:

un Alumno *es una* Persona  
un Docente *es una* Persona

```
public class Persona { ... }  
public class Alumno extends Persona { ... }  
public class Docente extends Persona { ... }
```





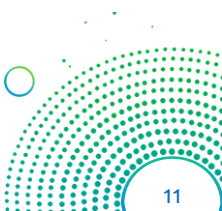
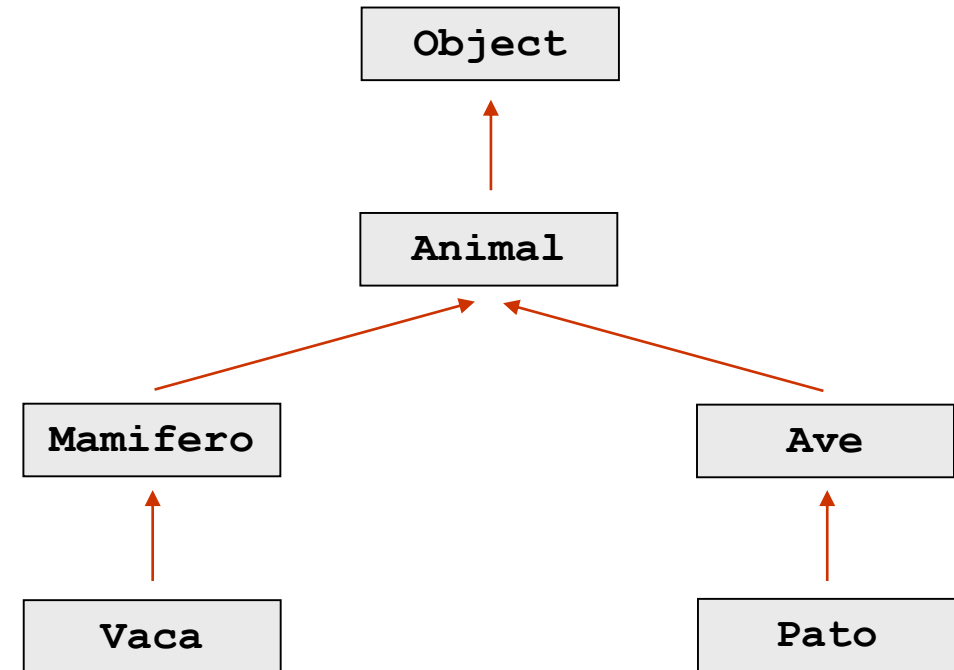
## 4.1.3 Relación “es un” o “es una”

- La herencia permite establecer una jerarquía de especialización mediante la relación “*es un*” o “*es una*”.

### Caso 2:

un Mamífero *es un* Animal  
un Ave *es un* Animal  
una Vaca *es un* Mamífero  
un Pato *es un* Ave

```
public class Animal { ... }  
public class Mamifero extends Animal { ... }  
public class Ave extends Animal { ... }  
public class Vaca extends Mamifero { ... }  
public class Pato extends Ave { ... }
```

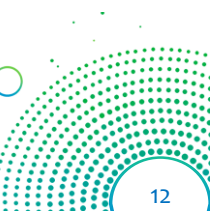




## 4.1.4 Uso de *super*

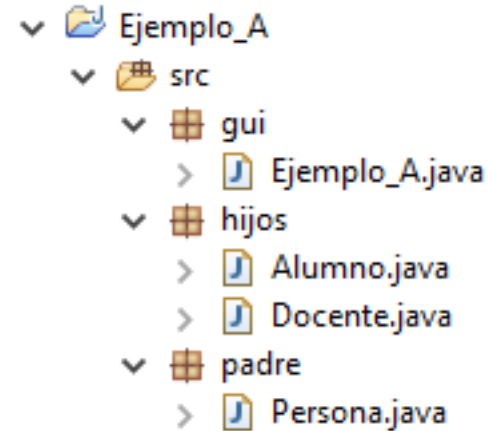
- El constructor de la clase **Hijo** utiliza la palabra **super** para invocar al constructor de la clase **Padre**.
- Esta instrucción tiene que ser la primera sentencia dentro del constructor de la clase **Hijo**.
- Por ejemplo:

```
public class Persona {  
    ...  
    public Persona(..., ...) {  
        ...  
    }  
    ...  
}  
public class Alumno extends Persona {  
    ...  
    public Alumno(..., ..., ..., ...) {  
        super(..., ...);  
        ...  
    }  
    ...  
}
```





# Ejemplo\_A



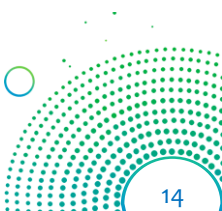
- Implementa la clase **Persona** en el *package* **padre** los atributos protegidos nombre, apellido y edad; un constructor que inicializa los atributos y el método *datosDeLaPersona()* que retorna en una cadena los datos completos de la persona incluyendo la generación de un correo electrónico.
- Implementa la clase **Alumno** en el *package* **hijos** que hereda información de la clase **Persona**, invoca al constructor de la clase **Persona** y declara además tres notas privadas. Adicionalmente implementa el método *datosDelAlumno()* que retorna en una cadena los datos completos del alumno incluyendo el promedio.
- Implementa la clase **Docente** en el *package* **hijos** que hereda información de la clase **Persona**, invoca al constructor de la clase **Persona** y declara además horas y tarifa como privados. Adicionalmente implementa el método *datosDelDocente()* que retorna en una cadena los datos completo del docente incluyendo el sueldo.
- En **Ejemplo\_A** declara tres objetos: de tipo **Persona**, **Alumno** y **Docente** respectivamente. Finalmente a través de tres métodos listado visualiza la información completa de cada objeto.



## 4.1.5 Uso de *super* en sobre escritura

- Se utiliza ***super*** también para acceder desde la clase **Hijo** a un elemento de la clase **Padre**, en caso que la clase **Hijo** cuente con un atributo o método igual en sintaxis.
- Por ejemplo:

```
public class Persona {  
    ...  
    protected int x = 10;  
    ...  
}  
public class Alumno extends Persona {  
    ...  
    private int x = 20;  
    public int y = super.x + x; // y = 30  
    ...  
}
```





## 4.1.6 Operador *instanceof*

- Se utiliza para determinar si el objeto es de la clase esperada.
- Por ejemplo:

```
void listado(Persona x) {  
    if (x instanceof Alumno) {  
        imprimir(">>> A L U M N O");  
        imprimir("clase Hijo");  
    }  
    else  
        if (x instanceof Docente) {  
            imprimir(">>> D O C E N T E");  
            imprimir("clase Hijo");  
        }  
        else {  
            imprimir(">>> P E R S O N A");  
            imprimir("clase Padre");  
        }  
}
```

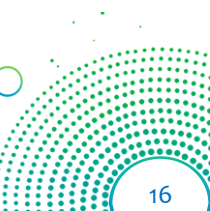




## 4.1.7 Polimorfismo

- El polimorfismo está presente cuando se realiza la llamada a un método de un objeto del que no se sabe su tipo hasta que el programa esté en ejecución. Al tener métodos sobrescritos, objetos de diferentes tipos pueden responder de forma diferente a la misma llamada, de tal forma que podemos escribir código de forma general sin preocuparnos del método concreto que se ejecutará en cada momento.
- Por ejemplo:

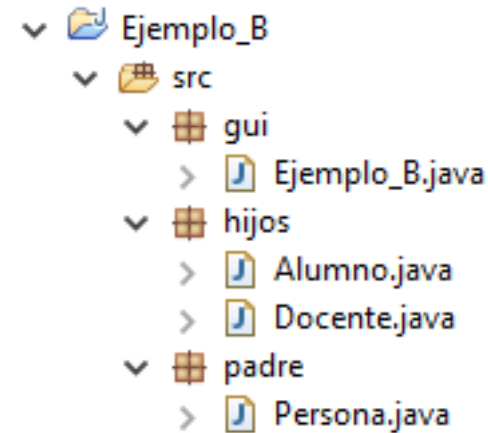
```
void listado(Persona x) {  
    if (x instanceof Alumno)  
        imprimir(">>> A L U M N O");  
    else  
        if (x instanceof Docente)  
            imprimir(">>> D O C E N T E");  
        else  
            imprimir(">>> P E R S O N A");  
    imprimir(x.datosCompleto());  
}
```







# Ejemplo\_B



- a) Cambia la sintaxis de los métodos que retornan los datos completos de cada clase por el de *datosCompletos()* y hace uso de *super* en las clases **hijos** para diferenciarlos.
- b) Utiliza un solo método listado que recibe como parámetro la referencia de la clase padre **Persona**, hace uso del operador *instanceof* para distinguir el tipo de objeto recibido y visualiza la información completa de los objetos.



# GRACIAS



## **SEDE MIRAFLORES**

Calle Díez Canseco Cdra 2 / Pasaje Tello  
Miraflores – Lima  
Teléfono: 633-5555

## **SEDE INDEPENDENCIA**

Av. Carlos Izaguirre 233  
Independencia – Lima  
Teléfono: 633-5555

## **SEDE BREÑA**

Av. Brasil 714 – 792  
(CC La Rambla – Piso 3)  
Breña – Lima  
Teléfono: 633-5555

## **SEDE TRUJILLO**

Calle Borgoño 361  
Trujillo  
Teléfono: (044) 60-2000

## **SEDE SAN JUAN DE LURIGANCHO**

Av. Próceres de la Independencia 3023-3043  
San Juan de Lurigancho – Lima  
Teléfono: 633-5555

## **SEDE LIMA CENTRO**

Av. Uruguay 514  
Cercado – Lima  
Teléfono: 419-2900

## **SEDE BELLAVISTA**

Av. Mariscal Oscar R. Benavides 3866 – 4070  
(CC Mall Aventura Plaza)  
Bellavista – Callao  
Teléfono: 633-5555

## **SEDE AREQUIPA**

Av. Porongoche 500  
(CC Mall Aventura Plaza)  
Paucarpata - Arequipa  
Teléfono: (054) 60-3535