HERENCIA EN JAVA

La herencia es un mecanismo que permite a una clase adquirir los atributos y métodos de otra clase. Por ejemplo, en una relación familiar: los hijos heredan características de sus padres. En programación, a esto se le llama una relación "es un" (por ejemplo, "un Coche es un Vehículo").

Esto sirve para reutilizar código y crear una jerarquía lógica entre clases.

El Concepto: Padre e Hijo

- **Superclase (Clase Padre):** Es la clase general que tiene las características y comportamientos comunes.
- **Subclase (Clase Hija):** Es la clase más específica que hereda de la clase padre y, además, puede tener sus propias características y comportamientos únicos.

Ejemplo Práctico: Vehículos

Vamos a modelar vehículos. Todos los vehículos tienen una marca y un modelo, y todos pueden avanzar.

1. Creamos la Clase Padre: `Vehiculo`

Esta es nuestra clase general. Define lo que cualquier vehículo tendrá.

```
class Vehiculo {
    String marca;
    String modelo;

    // Constructor para inicializar los atributos comunes
    Vehiculo(String marca, String modelo) {
        this.marca = marca;
        this.modelo = modelo;
    }

    // Un método que todos los vehículos pueden hacer
    void avanzar() {
        System.out.println("El vehiculo esta avanzando.");
    }
}
```

2. Creamos una Clase Hija: Coche

Un coche "es un" vehículo, pero tiene algo específico: un número de puertas. Usamos la palabra clave **`extends`** para indicar que `Coche` hereda de `Vehiculo`.

```
// 'Coche' hereda de 'Vehiculo' usando la palabra 'extends'
class Coche extends Vehiculo {
    int numeroDePuertas;

    // Constructor para el Coche
    Coche(String marca, String modelo, int puertas) {
        // 'super()' llama al constructor de la clase padre (Vehiculo)
        // para inicializar 'marca' y 'modelo'.
        super(marca, modelo);

        // Ahora inicializamos el atributo propio de Coche
        this.numeroDePuertas = puertas;
    }

    // Un método específico que solo tienen los coches
    void tocarBocina() {
        System.out.println("¡Beep, beep!");
    }
}
```

Puntos clave aquí:

- **`extends Vehiculo`**: Esta es la línea mágica. Le dice a Java que `Coche` obtiene todo lo que tiene `Vehiculo`.
- **`super(marca, modelo)`**: La clase hija **debe** llamar al constructor de su padre para que los atributos heredados (`marca` y `modelo`) se configuren correctamente. `super()` se encarga de eso.

3. Poniéndolo todo junto

Ahora, en una clase principal, veamos cómo funciona.

```
class Taller {
    public static void main(String[] args) {
        // Creamos un objeto de la clase hija 'Coche'
        Coche miCoche = new Coche("Toyota", "Corolla", 4);

        // Podemos usar los atributos heredados de Vehiculo
        System.out.println("Marca del coche: " + miCoche.marca);
        System.out.println("Modelo del coche: " + miCoche.modelo);
```

```
        // Y también podemos usar el atributo propio de Coche
        System.out.println("Numero de puertas: " +
miCoche.numeroDePuertas);

        System.out.println("--------------------");

        // El objeto 'miCoche' puede usar el método heredado de
Vehiculo...
        miCoche.avanzar();

        // ...y también puede usar su propio método.
        miCoche.tocarBocina();
    }
}
```

Resultado en la consola:

Marca del coche: Toyota

Modelo del coche: Corolla

Numero de puertas: 4

El vehiculo esta avanzando.

¡Beep, beep!

¿Cuándo se usa Super ?

Cuando no: Si la clase padre (`Vehiculo`) tiene un constructor sin parámetros (un constructor vacío), Java lo llamará por ti de forma automática y secreta (implícita).

Ejemplo: Aquí, `Vehiculo` tiene un constructor vacío.

```
class Vehiculo {
    String marca;

    // Constructor SIN parámetros
    Vehiculo() {
        this.marca = "Genérica";
        System.out.println("Constructor del Vehiculo llamado.");
    }
```

```
}
```

La clase hija Coche no necesita llamar a `super()` explícitamente.

```
// Archivo: Coche.java
class Coche extends Vehiculo {
    int numeroDePuertas;

    Coche(int puertas) {
        // Java inserta aquí una llamada invisible a super();
        this.numeroDePuertas = puertas;
        System.out.println("Constructor del Coche llamado.");
    }
}
```

Cuando creas un `new Coche(4)`, la salida será:

Constructor del Vehiculo llamado.

Constructor del Coche llamado.

Cuando si:

Si la clase padre (`Vehiculo`) solamente tiene constructores que requieren parámetros, estás obligado a llamar a `super()` explícitamente desde el constructor de la clase hija.

Ejemplo: Ahora, `Vehiculo` solo se puede crear si le das una marca.

```
// Archivo: Vehiculo.java
class Vehiculo {
    String marca;

    // El ÚNICO constructor requiere un parámetro
    Vehiculo(String marca) {
        this.marca = marca;
    }
}
```

Si intentas crear la clase Coche sin llamar a `super(...)`, tendrás un error de compilación. El compilador no sabe qué "marca" pasarle al constructor del `Vehiculo`.

Forma Correcta: Debes pasarle los datos que el padre necesita.

```
// Archivo: Coche.java
class Coche extends Vehiculo {
    int numeroDePuertas;

    Coche(String marca, int puertas) {
        // AHORA ES OBLIGATORIO: llamas al constructor del padre
        // y le pasas la información que necesita.
        super(marca);

        this.numeroDePuertas = puertas;
    }
}
```

Aquí le estás diciendo a Java: "Para construir este coche, primero construye un vehículo usando esta marca, y luego termina de construir el coche añadiéndole las puertas".

La Regla de Oro

- **Si la clase padre tiene un constructor sin parámetros, la llamada a `super()` es opcional (Java la hace por ti).**
- **Si la clase padre solo tiene constructores con parámetros, la llamada a `super(...)` es obligatoria.**

Ejemplo de ejercicio aplicado:

Figuras Geométricas

Este ejercicio es nuevo y te servirá de guía. La idea es modelar figuras geométricas. Todas las figuras tienen un color, pero cada una calcula su área de forma diferente.

Paso 1: Definir la Clase Padre (`Figura`)

Esta clase tendrá las características comunes a todas las figuras. En este caso, todas tendrán un color.

```
// Archivo: Figura.java
```

```java
// Clase Padre o Superclase
class Figura {
    String color;

    // Constructor para inicializar el color
    Figura(String color) {
        this.color = color;
    }

    // Un método que podría ser común, aunque lo dejaremos simple
    void mostrarColor() {
        System.out.println("El color de la figura es: " + this.color);
    }
}
```

Paso 2: Crear la Primera Clase Hija (`Rectangulo`)

Un rectángulo "es una" figura. Hereda el color, pero añade sus propias características: base y `altura`. También tendrá su propia forma de calcular el área.

```java
// Archivo: Rectangulo.java
// Clase Hija o Subclase
class Rectangulo extends Figura {
    double base;
    double altura;

    // Constructor del Rectángulo
    Rectangulo(String color, double base, double altura) {
        // 1. Llamamos al constructor del padre (Figura) para darle el
color
        super(color);

        // 2. Inicializamos los atributos propios del Rectángulo
        this.base = base;
        this.altura = altura;
    }

    // Método propio del Rectángulo
    double calcularArea() {
        return this.base * this.altura;
    }
}
```

Paso 3: Crear la Segunda Clase Hija (`Circulo`)

Un círculo también "es una" figura. Hereda el color, pero su atributo específico es el `radio`. Su fórmula para el área es diferente.

```
// Archivo: Circulo.java
// Otra Clase Hija
class Circulo extends Figura {
    double radio;

    // Constructor del Círculo
    Circulo(String color, double radio) {
        // 1. Llamamos al constructor del padre (Figura)
        super(color);

        // 2. Inicializamos el atributo propio del Círculo
        this.radio = radio;
    }

    // Método propio del Círculo (usamos Math.PI para el valor de Pi)
    double calcularArea() {
        return Math.PI * this.radio * this.radio;
    }
}
```

Paso 4: Poner todo en acción en la Clase Principal

Ahora creamos objetos de las clases hijas y vemos cómo pueden usar tanto sus métodos propios como los heredados.

```
// Archivo: Lienzo.java
// Clase principal para probar nuestro código
class Lienzo {
    public static void main(String[] args) {
        // Creamos un objeto Rectangulo
        Rectangulo miRectangulo = new Rectangulo("Azul", 10.0, 5.0);

        // Creamos un objeto Circulo
        Circulo miCirculo = new Circulo("Rojo", 3.0);

        System.out.println("--- Datos del Rectangulo ---");
        // Usamos el método heredado de Figura
```

```
        miRectangulo.mostrarColor();
        // Usamos el método propio de Rectangulo
        System.out.println("El area es: " +
miRectangulo.calcularArea());

        System.out.println("\n--- Datos del Circulo ---");
        // Usamos el método heredado de Figura
        miCirculo.mostrarColor();
        // Usamos el método propio de Circulo
        System.out.println("El area es: " + miCirculo.calcularArea());
    }
}
```

Resultado en la consola:

--- Datos del Rectangulo ---
El color de la figura es: Azul
El area es: 50.0
--- Datos del Circulo ---
El color de la figura es: Rojo
El area es: 28.274333882308138

TALLER A REALIZAR

1. **Sistema de Biblioteca (Evolución):**
    - **Clase Padre: `Publicacion` (con atributos `titulo`, `autor`).**
    - **Clases Hijas: `Libro` (que hereda de `Publicacion` y añade `numeroPaginas`) y `Revista` (que hereda de `Publicacion` y añade `edicion`).**
    - **Objetivo: Simular que un `Miembro` puede tomar prestado tanto libros como revistas, demostrando que ambos "son un" tipo de publicación.**
2. **Cuenta Bancaria (Evolución):**
    - **Clase Padre: `CuentaBancaria` (con `numeroDeCuenta`, `saldo`, `depositar`, `retirar`).**
    - **Clases Hijas: `CuentaDeAhorro` (que hereda y añade un método `calcularInteres()`) y `CuentaCorriente` (que hereda y añade un atributo `limiteSobregiro`).**
    - **Objetivo: Crear un `Cliente` que pueda tener ambos tipos de cuenta y realizar operaciones específicas en cada una.**
3. **Gestión de Mascotas (Evolución):**
    - **Clase Padre: `Mascota` (con `nombre`, `especie`, `hacerSonido()`).**

- **Clases Hijas: `Perro` (que hereda y añade el método `traerPalo()`) y `Gato` (que hereda y añade el método `ronronear()`).**
- **Objetivo: Un `Dueño` adopta un perro y un gato, y puede pedirles que realicen tanto las acciones comunes (`hacerSonido`) como las específicas de su especie.**

4. **Inscripción a Cursos (Evolución):**
   - **Clase Padre: `Persona` (con `nombre`, `id`).**
   - **Clases Hijas: `Estudiante` (que hereda de `Persona` y añade una lista de `cursosInscritos`) y `Profesor` (que hereda de `Persona` y añade una lista de `cursosAsignados`).**
   - **Objetivo: Crear objetos de estudiantes y profesores, demostrar que ambos tienen nombre e ID, pero solo los estudiantes se pueden inscribir a cursos y solo los profesores pueden impartirlos.**
5. **Pedido en Pizzería (Evolución):**
   - **Clase Padre: `Producto` (con `nombre`, `precio`).**
   - **Clases Hijas: `Pizza` (que hereda de `Producto` y añade `ingredientes`) y `Bebida` (que hereda de `Producto` y añade `tamañoEnMl`).**
   - **Objetivo: Un `Pedido` ahora puede contener tanto pizzas como bebidas. El método `calcularTotal()` debe funcionar para sumar el precio de todos los productos, sin importar de qué tipo sean.**