



## Taller: Uso de **super** en Java

### Objetivo del Taller:

En este taller, los estudiantes aprenderán el propósito y el uso correcto de la palabra clave **super** en Java. Se explicará cómo se utiliza **super** para acceder a miembros y métodos de la clase base, así como para invocar constructores de la clase base. Se proporcionarán ejemplos prácticos y ejercicios para reforzar estos conceptos y evitar errores comunes.

### Temario:

- Contextualización y Definición del Uso de **super****
  - ¿Qué es **super** en Java?
  - ¿Cómo se utiliza y cuál es su propósito?
- Objetivos del Uso de **super****
  - Acceso a miembros y métodos de la clase base.
  - Invocación de constructores de la clase base.
- Cuándo Usar y Cuándo No Usar **super****
  - Casos recomendados y no recomendados.
- Sintaxis del Uso de **super****
  - Uso de **super** para acceder a métodos y atributos de la clase base.
  - Uso de **super** para invocar constructores de la clase base.
- Ejemplos de Uso Correcto**
  - Ejemplos prácticos que muestren la invocación de métodos y constructores.
- Ejemplos de Uso Incorrecto con Errores de Compilación**
  - Ejemplos que violan las reglas del uso de **super** y no compilan.
- Ejemplos de Uso Incorrecto sin Generar Error de Compilación**
  - Ejemplos que son malas prácticas pero que no generan errores.
- Ejercicios Propuestos**
  - Ejercicios para poner en práctica el uso correcto e incorrecto de **super**.

## 1. Contextualización y Definición del Uso de **super**

### ¿Qué es **super** en Java?

En Java, la palabra clave **super** se utiliza para referirse a la clase base (también llamada clase padre) desde una clase derivada. **super** permite:

- Acceder a los métodos y atributos de la clase base que han sido heredados.
- Invocar constructores de la clase base desde el constructor de la clase derivada.

### Propósito de **super**:

**super** se utiliza principalmente en el contexto de herencia, cuando se necesita hacer referencia explícita a miembros o métodos de la clase base, o invocar un constructor específico de la clase base desde la clase derivada.

## 2. Objetivos del Uso de **super**

### Objetivos Principales:

- Acceder a Métodos y Atributos de la Clase Base:** Cuando la clase derivada sobrescribe un método de la clase base, **super** permite invocar el método de la clase base para reutilizar su comportamiento.
- Invocar Constructores de la Clase Base:** **super** permite a la clase derivada invocar el constructor de la clase base, lo cual es esencial para inicializar los miembros heredados correctamente.

### Ejemplo Simple de Uso de **super** para Invocar un Constructor:

```
public class Animal {
    protected String especie;

    public Animal(String especie) {
        this.especie = especie;
    }
}
```



```
public class Perro extends Animal {
    private String raza;

    public Perro(String especie, String raza) {
        super(especie); // Llama al constructor de la clase base
        this.raza = raza;
    }
}
```

### 3. Cuándo Usar y Cuándo No Usar **super**

#### Cuándo Usar **super**:

- Cuando se necesita reutilizar el comportamiento de la clase base en un método sobrescrito.
- Cuando se necesita invocar el constructor de la clase base para inicializar atributos heredados.
- Cuando se requiere acceso explícito a miembros o métodos de la clase base que han sido ocultados o sobrescritos por la clase derivada.

#### Cuándo No Usar **super**:

- En métodos que no sobrescriben los métodos de la clase base.
  - Cuando no se necesita invocar el constructor de la clase base explícitamente (Java invoca implícitamente el constructor por defecto de la clase base si no se utiliza **super()**).
  - En contextos donde no hay herencia directa.
- 

### 4. Sintaxis del Uso de **super**

#### Uso de **super** para Acceder a Métodos y Atributos de la Clase Base:

```
public class Empleado {
    protected String nombre;

    public void mostrarNombre() {
        System.out.println("Nombre: " + nombre);
    }
}

public class Gerente extends Empleado {
    private String departamento;

    @Override
    public void mostrarNombre() {
        super.mostrarNombre(); // Llama al método de la clase base
        System.out.println("Departamento: " + departamento);
    }
}
```

#### Uso de **super** para Invocar Constructores de la Clase Base:

```
public class Vehiculo {
    protected String marca;

    public Vehiculo(String marca) {
        this.marca = marca;
    }
}

public class Coche extends Vehiculo {
    private int numeroDePuertas;

    public Coche(String marca, int numeroDePuertas) {
```



```
    super(marca); // Llama al constructor de la clase base
    this.numeroDePuertas = numeroDePuertas;
}
}
```

## 5. Ejemplos de Uso Correcto

### Ejemplo Correcto 1: Invocar el Constructor de la Clase Base

```
public class Animal {
    protected String especie;

    public Animal(String especie) {
        this.especie = especie;
    }

    public void mostrarEspecie() {
        System.out.println("Especie: " + especie);
    }
}

public class Gato extends Animal {
    private String color;

    public Gato(String especie, String color) {
        super(especie); // Llama al constructor de la clase base
        this.color = color;
    }

    @Override
    public void mostrarEspecie() {
        super.mostrarEspecie(); // Llama al método de la clase base
        System.out.println("Color: " + color);
    }
}
```

## 6. Ejemplos de Uso Incorrecto con Errores de Compilación

### Ejemplo Incorrecto 1: Intentar Usar **super** Fuera del Contexto de una Clase Derivada

```
public class Planta {
    private String tipo;

    public Planta(String tipo) {
        this.tipo = tipo;
    }
}

public class Main {
    public static void main(String[] args) {
        // Error: No se puede usar `super` aquí porque Main no es una clase derivada de Planta
        super.tipo = "Orquídea"; // Error de compilación
    }
}
```

**Explicación:** La palabra clave **super** solo se puede utilizar en el contexto de una clase derivada. El intento de usarla fuera de una relación de herencia directa genera un error de compilación.

## 7. Ejemplos de Uso Incorrecto sin Generar Error de Compilación



### Ejemplo Incorrecto 2: Uso Innecesario de `super`

```
public class Vehiculo {
    protected String tipo;

    public void mostrarTipo() {
        System.out.println("Tipo de vehículo: " + tipo);
    }
}

public class Bicicleta extends Vehiculo {
    @Override
    public void mostrarTipo() {
        super.mostrarTipo(); // Uso innecesario si el método no está sobrescrito o modificado
        System.out.println("Este es un tipo de bicicleta.");
    }
}
```

**Explicación:** Aunque el uso de `super` aquí no genera un error de compilación, es innecesario, ya que el método `mostrarTipo` de la clase base no ha sido modificado.

---

## 8. Ejercicios Propuestos

### Ejercicio 1: Clase `Persona` y Clase `Empleado`

1. Define una clase `Persona` con los atributos `nombre` y `edad`, y un método `mostrarDetalles`.
2. Define una clase `Empleado` que herede de `Persona` y agregue un atributo `departamento`.
3. En la clase `Empleado`, sobrescribe el método `mostrarDetalles` para mostrar también el departamento utilizando `super`.

### Ejercicio 2: Clase `Animal` y Clase `Pez`

1. Define una clase `Animal` con un atributo `especie` y un método `mostrarEspecie`.
2. Define una clase `Pez` que herede de `Animal` y agregue un atributo `tipoDeAgua`.
3. Utiliza `super` para invocar el constructor y el método de la clase base desde `Pez`.

### Ejercicio 3: Uso Incorrecto de `super`

1. Intenta utilizar `super` en un contexto que no sea una clase derivada y observa los errores de compilación.
  2. Intenta invocar un atributo privado de la clase base utilizando `super` y discute por qué ocurre un error.
- 

## Conclusión del Taller:

Este taller ha proporcionado una comprensión clara del uso de `super` en Java, explicando su propósito y cómo permite acceder a métodos, atributos y constructores de la clase base. `super` es una herramienta fundamental para la reutilización del código en un contexto de herencia.

### Puntos Clave a Recordar:

- La palabra clave `super` se utiliza para referirse a la clase base desde una clase derivada.
- Permite acceder a métodos y atributos de la clase base que han sido sobrescritos o modificados.
- Facilita la invocación de constructores de la clase base desde el constructor de la clase derivada.