




**UNIVERSIDAD TECNOLÓGICA DEL PERÚ**  
**Facultad de Ingeniería**

# **Programacion Orientada a Objetos**

**Sesión 5: Sustitución de métodos de la superclase**

# Agenda


- 
- Introducción a la sustitución de métodos
  - ¿Por qué sustituir métodos?
  - Cómo sustituir métodos en la superclase
  - Ejemplos de sustitución de métodos


# Logros

- Entender la importancia de la sustitución de métodos en la programación orientada a objetos.
- Saber cómo y cuándo sustituir un método de la superclase.
- Identificar casos de uso para la sustitución de métodos.



## Conocimientos previos

- Conocimientos básicos de programación orientada a objetos.
  - Entender la relación entre superclases y subclases.
  - Conocer la herencia y cómo funciona.
- 




# Introducción a la sustitución de métodos

- En programación orientada a objetos, las subclases pueden heredar métodos de la superclase.
- A veces, es necesario modificar o extender la funcionalidad de un método heredado.
- Esto se logra mediante la sustitución de métodos.



## ¿Por qué sustituir métodos?

- Para adaptar o mejorar la funcionalidad heredada.
  - Para implementar comportamientos específicos en la subclase.
  - Para corregir errores o limitaciones en el método heredado.
- 

# Cómo sustituir métodos en la superclase

- Definir un método en la subclase con el mismo nombre que el método en la superclase.
- El método en la subclase ocultará o sobrescribirá el método en la superclase.
- Se puede usar la palabra clave 'super' para llamar al método original de la superclase si es necesario.



## Ejemplos de sustitución de métodos

La sustitución de métodos (también conocida como "overriding" en inglés) es un concepto fundamental en la programación orientada a objetos. En Java, esto permite que una subclase proporcione una implementación específica de un método que ya está definido en su superclase.

Vamos a ver un ejemplo paso a paso:

### 1. Definición de la superclase

Supongamos que tenemos una superclase llamada `Animal` que tiene un método `sonido()`:



```
class Animal {  
    void sonido() {  
        System.out.println("El animal hace un sonido");  
    }  
}
```

## 2. Creación de una subclase

Ahora, vamos a crear una subclase llamada Perro que hereda de Animal. Queremos que el perro tenga su propio sonido, por lo que vamos a sobrescribir el método sonido():

```
class Perro extends Animal {  
    void sonido() {  
        System.out.println("El perro ladra");  
    }  
}
```

- 3. Uso de la sustitución de métodos

Ahora, si creamos un objeto de tipo Perro y llamamos al método sonido(), se llamará al método sobrescrito de la subclase Perro en lugar del método de la superclase Animal:

```
public class TestAnimales {  
    public static void main(String[] args) {  
        Animal miPerro = new Perro();  
        miPerro.sonido(); // Salida: El perro ladra  
    }  
}
```

- Notas importantes:

La palabra clave `@Override` se puede usar para indicar que estamos sobrescribiendo un método. Esto es útil porque el compilador verificará si realmente estamos sobrescribiendo un método de la superclase. Si no es así, se generará un error.

```
class Perro extends Animal {  
    @Override  
    void sonido() {  
        System.out.println("El perro ladra");  
    }  
}
```

Para que un método en la superclase pueda ser sobrescrito por la subclase, debe ser accesible (no ser `private`) y no estar marcado como `final`.



# Conclusiones

- La sustitución de métodos es una herramienta poderosa en programación orientada a objetos.
  - Permite a las subclasses adaptar o extender funcionalidades heredadas.
  - Es esencial usarla con cuidado para mantener la coherencia y evitar errores.
- 