## Polimorfismo

En Java, el **polimorfismo** (del griego "muchas formas") es un principio fundamental de la programación orientada a objetos que permite que objetos de diferentes clases respondan al mismo mensaje (invocación de método) de maneras específicas para cada clase. En esencia, permite tratar a objetos de distintas clases como si fueran del mismo tipo, lo que aporta una gran flexibilidad y reutilización al código.

La implementación del polimorfismo en Java se apoya fuertemente en los conceptos de **herencia** e **interfaces**. Una variable de un tipo de superclase puede hacer referencia a un objeto de cualquiera de sus subclases, y el método que se ejecutará será el correspondiente al tipo real del objeto en tiempo de ejecución.

### Tipos de polimorfismo

#### Polimorfismo en Tiempo de Compilación (Estático)

También conocido como **enlace estático**, se resuelve durante la compilación del programa. La forma más común de polimorfismo en tiempo de compilación es la **sobrecarga de métodos** (*method overloading*).

**Sobrecarga de Métodos:** La sobrecarga de métodos consiste en definir varios métodos con el mismo nombre dentro de la misma clase, pero con diferentes listas de parámetros (ya sea por el número de parámetros, el tipo de los parámetros o ambos).

#### Polimorfismo en Tiempo de Ejecución (Dinámico)

También conocido como **enlace dinámico**, se resuelve durante la ejecución del programa. La forma principal de lograr el polimorfismo en tiempo de ejecución es la **sobrescritura de métodos** (*method overriding*).

**Sobrescritura de Métodos:** La sobrescritura de métodos ocurre cuando una clase hija (subclase) proporciona una implementación específica para un método que ya está definido en su clase padre (superclase). Para que sea una sobrescritura, el nombre del método, los parámetros y el tipo de retorno deben ser los mismos que en el método de la superclase. Se recomienda usar la anotación **@Override** para asegurar que se está sobrescribiendo correctamente un método.

#### 

### Uso del polimorfismo

* **Reutilización de código:** Permite que las clases hijas reutilicen métodos y atributos de la clase padre.
* **Flexibilidad y extensibilidad:** Facilita la adición de nuevas clases al sistema sin necesidad de modificar el código existente que trabaja con la jerarquía de clases. En el ejemplo anterior, se podría añadir una clase Vaca que herede de Animal sin cambiar el código del main.
* **Código más limpio y legible:** Reduce la necesidad de sentencias condicionales (if-else o switch) para verificar el tipo de un objeto antes de invocar un método.
* **Desacoplamiento:** Promueve un bajo acoplamiento entre las clases, lo que hace que el sistema sea más fácil de mantener y evolucionar.