

# Polimorfismo por Sobrescritura

Material de clase elaborado por Sandra Victoria Hurtado Gil

Una de las características importantes de los objetos, y en general de la Programación Orientada a Objetos (POO), es la capacidad de comportarse diferente ante un mismo mensaje, dependiendo de los parámetros que recibe o del tipo de objeto que sea. Esto se conoce como **polimorfismo**.

Esta capacidad de comportarse diferente puede darse en dos casos:

- Cuando la información que se le da al objeto para solicitarle el servicio cambia. Es decir, se envían diferente tipo y cantidad de parámetros a un método. Esto se conoce como **polimorfismo por sobrecarga**.
- Cuando se pide el mismo servicio a diferentes objetos, y cada uno de ellos puede hacerlo de forma diferente. Esto se conoce como **polimorfismo por sobrescritura**.

El polimorfismo por sobrescritura se presenta cuando se tienen objetos que son especialización (hijos) de otro, y cada uno se comporta de forma diferente cuando se les pide el mismo método.

Por ejemplo, si se le pide a un carro de carreras que acelere, lo hace de manera casi inmediata y alcanza una gran velocidad, pero si se le pide a un carro clásico que acelere, lo hará lentamente y no alcanzará una gran velocidad.

En este caso es el mismo servicio ("acelerar") solicitado a dos objetos que son carros. La diferencia está en que, aunque ambos son carros, uno es un carro de carreras y el otro es un carro clásico, es decir, son especializaciones o tipos de carro.

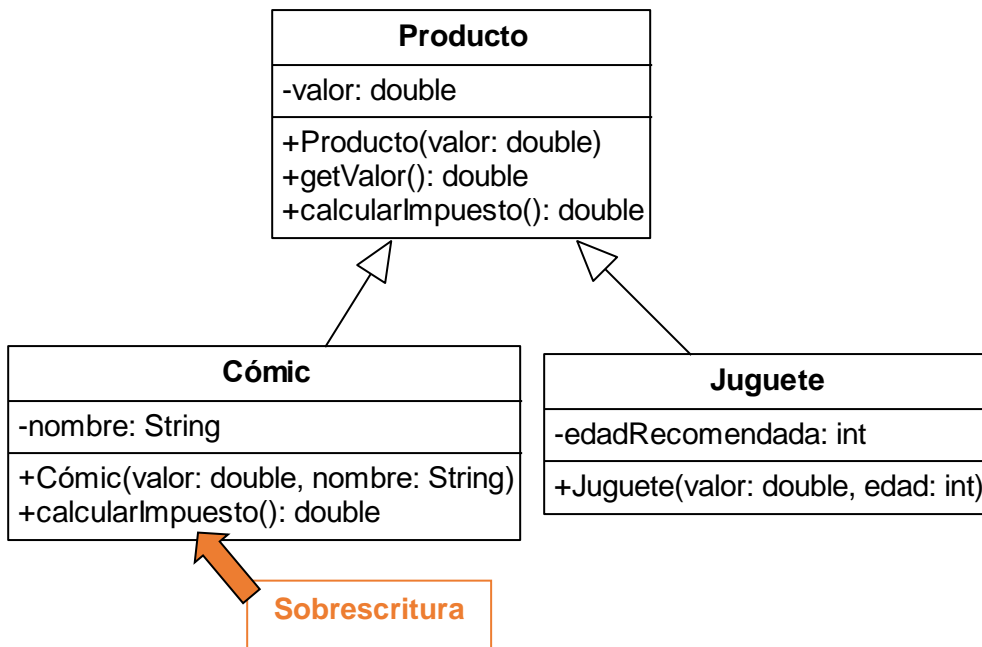
## Implementación en Java

En Java el polimorfismo por sobrescritura se logra de tres formas:

- Cuando se tiene un método definido en la clase padre y las clases hijas definen de nuevo este método para darle un comportamiento diferente al que estaba en la clase padre. En este caso las hijas no están obligadas a definir de nuevo el método de la clase padre.
- Cuando se tiene un método abstracto en una clase padre (también abstracta) y las hijas deben definir (por primera vez) el método, dándole un cuerpo. En este caso las hijas están obligadas a definir el método.
- Cuando se tiene un método abstracto en una interfaz y las clases que la implementan deben definir (por primera vez) el método, dándole un cuerpo. En este caso estas clases también están obligadas a definir el método.

En este material solo se presentará el primer caso.

Por ejemplo, se tiene el siguiente diagrama de clases de una tienda de variedades:



En la clase **Producto** se define un método llamado “calcularImpuesto”, que calcula el impuesto de venta que se debe aplicar a los productos que se venden en la tienda. Normalmente el impuesto corresponde al 14 % del valor del producto. Sin embargo, los comics se consideran un producto de lujo, por lo que pagan 16 % de impuesto.

En la clase **Producto** el método “calcularImpuesto” está definido para calcular el impuesto con el 14 % y todas sus clases hijas lo heredan. Sin embargo, la clase hija **Cómic**, como calcula diferente el impuesto, define de nuevo el método (**con el mismo encabezado**) y se queda con esta nueva definición.

De acuerdo con el diagrama anterior:

- La clase **Cómic** **sobrescribe** el método
- La clase **Juguete** NO sobrescribe el método, es decir, se queda con el mismo que hereda de la clase padre porque así le sirve.

El código para este diagrama es:

```

/**
 * Elemento que se vende en una tienda de variedades,
 * y al cual se le calcula el impuesto a partir del valor.
 * @version 1.0
 */
public class Producto {
    private double valor;

    public Producto(double valor) {
        this.valor = valor;
    }

    public double getValor() {
        return valor;
    }

    /**
     * Calcula el impuesto del producto.
     * Normalmente el impuesto es el 14% del valor del producto.
     * @return el impuesto correspondiente para el producto
     */
    public double calcularImpuesto() {
        double impuesto = valor * 0.14;
        return impuesto;
    }
} // Fin clase Producto

```

```

/**
 * Elemento para jugar, que se vende en la tienda de variedades
 * @version 1.0
 */
public class Juguete extends Producto {
    private int edadRecomendada;

    public Juguete(double valor, int edad) {
        super(valor);
        this.edadRecomendada = edad;
    }
} // Fin clase Juguete

```


```

/**
 * Revista de historietas que se vende en la tienda
 * @version 1.0
 */
public class Comic extends Producto {
    private String nombre;

    public Comic(double valor, String nombre) {
        super(valor);
        this.nombre = nombre;
    }

    /**
     * Calcula el impuesto, que en este caso es el 16% del valor
     * (por ser un bien de lujo).
     * @return el valor del impuesto que se debe pagar
     */
    @Override
    public double calcularImpuesto()
    {
        double impuesto = getValor()*0.16;
        return impuesto;
    }
} // Fin clase Comic

```



Anotación que indica  
que el método se  
sobrescribe

La anotación `@Override` ayuda a prevenir errores porque permite que el compilador de Java revise que el encabezado del método coincida con el que está definido en la clase papá.

Ahora, se tiene el siguiente código correspondiente a la clase `Tienda`, con el método `main`, donde se crean dos productos (un juguete y un cómic) con el mismo valor y se calcula el impuesto de cada uno:

```

/**
 * Lugar donde se venden productos variados como juguetes y cómics
 * @version 1.0
 */
public class Tienda {
    public static void main(String[] args) {
        Producto productoA = new Comic(1000, "Superhéroes");
        Producto productoB = new Juguete(1000, 3);
        System.out.println("Impuesto A: " + productoA.calcularImpuesto());
        System.out.println("Impuesto B: " + productoB.calcularImpuesto());
    }
}

```

La salida es:

```

Impuesto A: 160.0
Impuesto B: 140.0

```

Puede verse como los objetos, aunque tienen el mismo valor y se les llama el mismo método, se comportan de forma diferente, por la sobrescritura.

Incluso aunque los objetos son referenciados por una variable de la clase papá, Java utiliza el método que corresponda a cada objeto. Es como “engañar” a Java, porque el método está definido en la clase papá y por lo tanto se puede usar con la variable de tipo Producto, pero como el objeto es realmente un Cómic, entonces se usa la nueva definición que estableció esta clase. En cuanto al objeto Juguete, se usa el método definido en la clase padre, dado que lo hereda y no lo cambia.