



CICLO 2

[FORMACIÓN POR CICLOS]


Herencia en **JAVA**



Ingeni@
Soluciones TIC



UNIVERSIDAD
DE ANTIOQUIA
Facultad de Ingeniería

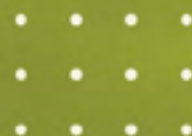


Uno de los mecanismos más potentes de la programación orientada a objetos es la herencia. La **herencia** permite que una clase “herede” los métodos y atributos de otra clase. Esto es, una clase, a menudo denominada clase hija o subclase, hereda de una clase padre o superclase sus métodos y atributos. Así, la clase hija tendrá los métodos y atributos de la clase padre, más aquellos que se definan dentro de ella misma, constituyendo esto uno de los mecanismos más efectivos para la reutilización de código existente.

Cabe señalar, que una clase puede tener una clase padre, y esta a su vez otra clase padre, y así sucesivamente. Cuando esto se da, se presenta lo que denominamos una **jerarquía de clases**. Además, en Java todas las clases heredan directa o indirectamente de la clase Object. Es decir, si bien en el código fuente de una clase no será necesario definirlo explícitamente, todas las clases dentro del lenguaje tienen a la clase Object como clase padre.

Visto lo anterior, hay algunos aspectos a tener en cuenta:

- Para ser más exactos, la subclase (clase hija) hereda los métodos y atributos públicos (public) y protegidos (protected) de la superclase (clase padre). Esto implica que la clase hija sigue sin tener acceso directo a los métodos y atributos privados de la clase padre. En el caso de los atributos privados, se deberán acceder mediante los *getters* y los *setters* existentes.
- Los métodos y atributos heredados se podrán usar como si fueran propios, usando la palabra clave *this*, a menos que se estén sobrescribiendo (como veremos más abajo).
- Una subclase puede “redefinir” los métodos de su superclase. Es decir, de ser necesario, podemos cambiar cómo funciona un método existente en la clase padre al realizar la herencia en la clase hija (se sobrescribe el método existente, tal como veremos más abajo).
- Los constructores de la superclase no se heredan, pero se pueden llamar desde los constructores de la subclase.



Finalmente, cabe señalar que el tipo de herencia que se realiza en el lenguaje Java es la **herencia simple**. En la programación orientada a objetos en general existen dos tipos de herencia, herencia simple y herencia múltiple. La herencia múltiple, tal como es posible en Python, permite que una clase tenga más de una clase padre, es decir, que herede de más de una superclase. Por otro lado, la herencia simple, tal como se da en Java, sólo permite que una clase tenga una sola clase padre.

Implementación de la herencia en Java

Para indicar que una clase hereda de otra en Java, se utiliza la palabra clave `extends` después del nombre de la clase, tal como se puede ver a continuación.

```
package co.edu.udea.udea_ruta2_ciclo2.poo;  
  
public class BicicletaMontania extends Bicicleta {  
  
}
```

Siguiendo con el ejemplo de lecturas anteriores, arriba tenemos la clase `BicicletaMontania` que hereda de la clase (existente) `Bicicleta`. Esto quiere decir que esta clase hereda los métodos y atributos de su clase padre, pudiendo acceder además a sus constructores y con la posibilidad de agregar nuevas funcionalidades. A continuación, vamos a agregar un atributo, de tal manera que la bicicleta de montaña funcione con 12 cambios, y los métodos que nos permitirán subir y bajar de cambio, así como conocer el cambio actual.



```
package co.edu.udea.udea_ruta2_ciclo2.poo;

public class BicicletaMontania extends Bicicleta {

    private int cambio;

    public void subirCambio() {
        if (this.cambio < 12) {
            cambio++;
        }
    }

    public void bajarCambio() {
        if (this.cambio > 1) {
            cambio--;
        }
    }

    public int getCambio() {
        return this.cambio;
    }
}
```

Así las cosas, tenemos una clase BicicletaMontania con los métodos subirCambio, bajarCambio y getCambio. Pero, además, también cuenta con los métodos que hereda de Bicicleta, estos son: getMarca, setMarca, getColor, setColor, getVelocidad, pedalear y frenar (en sus diferentes variantes).

Recordemos que, además se reutilizar el código de la superclase, en una subclase también podemos **sobreescribir** el código de uno o varios métodos. A continuación, sobreescribimos el método pedalear. Dado que la bicicleta ya cuenta con cambios, lo más lógico es que la velocidad de la misma aumente de manera proporcional al cambio actual.



```
package co.edu.udea.udea_ruta2_ciclo2.poo;

public class BicicletaMontania extends Bicicleta {


    private int cambio;

    public void subirCambio() {
        if (this.cambio < 12) {
            cambio++;
        }
    }

    public void bajarCambio() {
        if (this.cambio > 1) {
            cambio--;
        }
    }

    public int getCambio() {
        return this.cambio;
    }

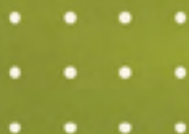
    @Override
    public void pedalear(double aceleracion) {
        double aceleracionConCambios = aceleracion *
        (this.cambio/6);
        super.pedalear(aceleracionConCambios);
    }
}
```

Si nos fijamos en el nuevo método pedalear, encontraremos varias particularidades:

- Justo encima del método encontramos la anotación @Override. Esta anotación se debe poner, siempre, encima de los métodos que sobrescriben métodos de una clase padre. Por lo general los IDEs nos indican cuando es necesario.
- El método tiene el mismo nombre, y el mismo tipo y número de parámetros que el método pedalear de la clase Bicicleta. Al sobrescribir dicho método, cuando creamos un objeto de la clase BicicletaMontania, el método pedalear funcionará con cambios. El método original, en la clase Bicicleta, seguirá funcionando como siempre.
- Al no ser posible acceder directamente al atributo velocidad de Bicicleta (ya que es privado), es necesario usar el método pedalear de dicha clase. Para acceder al método de la clase padre que estamos sobrescribiendo, se usa la palabra clave super, de manera similar a como usamos la palabra clave this para métodos de la clase actual.

Finalmente, vamos a agregar constructores a nuestra clase BicicletaMontania.



```
package co.edu.udea.udea_ruta2_ciclo2.poo;

public class BicicletaMontania extends Bicicleta {

    private int cambio;

    public BicicletaMontania() {
        super();
        this.cambio = 6;
    }

    public BicicletaMontania(int cambio, String marca,
String color, double velocidadInicial) {
        super(marca, color, velocidadInicial);
        if(cambio > 0 && cambio <= 12) {
            this.cambio = cambio;
        } else {
            this.cambio = 6;
        }
    }

    public void subirCambio() {
        if (this.cambio < 12) {
            cambio++;
        }
    }

    public void bajarCambio() {
        if (this.cambio > 1) {
            cambio--;
        }
    }

    public int getCambio() {
        return this.cambio;
    }

    @Override
    public void pedalear(double aceleracion) {
        double aceleracionConCambios = aceleracion *
(this.cambio/6.0);
        super.pedalear(acceleracionConCambios);
    }
}
```

A la clase anterior se le agregaron dos constructores: uno sin parámetros y otro con los parámetros correspondientes a los atributos de Bicicleta y BicicletaMontania:

- En el primer caso, se llama el constructor sin parámetros de la superclase Bicicleta usando la palabra clave super, para luego asignar un valor por defecto al atributo cambio.
- En el segundo caso, se llama el constructor de la superclase Bicicleta con los parámetros correspondientes a todos sus atributos usando la palabra clave super. Luego, se asigna un valor al atributo cambio (recibido mediante el parámetro respectivo), siempre que este cumpla la restricción de estar entre 1 y 12.

Una vez se ha creado la clase BicicletaMontania, podremos crear objetos a partir de esta, e incluso, podremos definir clases hijas de esta, de ser necesario¹.

```
package co.edu.udea.udea_ruta2_ciclo2.poo;

public class PruebaHerencia {

    public static void main(String[] args) {
        BicicletaMontania miTrochera = new
        BicicletaMontania();
        System.out.println("Mi bicicleta es de marca " +
        miTrochera.getMarca()
            + ", de color " + miTrochera.getColor()
            + ", con una velocidad de " + miTrochera.
            getVelocidad()
            + " y está en el cambio " + miTrochera.
            getCambio());
        miTrochera.setMarca("Specialized");
        miTrochera.setColor("Rojo");
        miTrochera.subirCambio();
        miTrochera.subirCambio();
        miTrochera.subirCambio();
        miTrochera.pedalear(5);
        miTrochera.pedalear(5);
        System.out.println("Mi bicicleta es de marca " +
        miTrochera.getMarca()
            + ", de color " + miTrochera.getColor()
            + ", con una velocidad de " + miTrochera.
            getVelocidad()
            + " y está en el cambio " + miTrochera.
            getCambio());
    }
}
```

²Los ejemplos anteriores se pueden ver acá: https://github.com/leonjara-millo/udea_ruta2_ciclo2/tree/main/main/java/co/edu/udea/udea_ruta2_ciclo2/poo