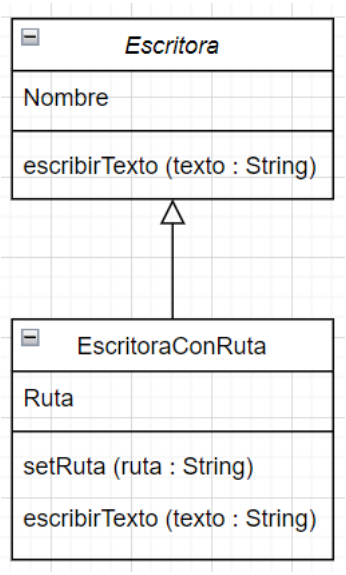
Herencia

En la Programación Orientada a Objetos, la **Herencia** es un mecanismo que permite crear **nuevas clases a partir de otras** ya existentes, que se suponen comprobadas y correctas. Se evita así el problema de tener que andar rediseñando, reescribiendo y modificando una clase que ya existente (y funciona) o duplicando código innecesariamente.



Por ejemplo, tenemos una clase **Escritora** que permite escribir texto un fichero. Esta clase funciona perfectamente, pero nuestro programa necesita una clase que además permita elegir la ruta donde se creará el fichero. En lugar de escribir **EscritoraConRuta** desde cero, podemos utilizar la herencia de clases.

Al hacer que **EscritoraConRuta** herede de **Escritora**, obtiene todo el comportamiento (métodos) y los atributos (variables) de su superclase. Dispondrá del atributo nombre y del método escribirTexto () como si fuese **una copia** de su superclase. Pero eso no impide que modifiquemos cosas en la nueva clase. Si te fijas, **EscritoraConRuta** tiene un atributo ruta y un método setRuta () que su superclase no tiene. Adicionalmente, el hecho de que en el esquema aparezca el método escribirTexto () significa que, aunque lo tenga por herencia, lo hemos **reescrito** y su comportamiento será diferente al de **Escritora**.

A la clase de la que derivan otras clases que extienden su funcionalidad se la denomina clase base, **clase padre** o superclase; mientras que las clases derivadas se denominan **clases hijas** o subclases.

En el caso de Java, TODAS las clases heredan SIEMPRE de otras clases. Esto crea una estructura de árbol jerárquica en la que la clase **Object** se encuentra en la raíz. Cuando creas una nueva clase del tipo que sea, se te plantean dos opciones: no indicar nada, lo que hace que tu clase herede de Object; o indicar específicamente cuál es tu superclase.

En Java NO ES POSIBLE que una clase herede de dos superclases a la vez.

Herencia sencilla - Extends

Para indicar que una clase hereda de otra, basta con indicarlo con la palaba reservada **extends**. Mira el ejemplo siguiente. La clase Escritora dispone de un atributo nombre, un método escribirTexto () y un getter-setter.

```
public class Escritora {

    private String nombre = null;

    public void escribirTexto (String texto) {
        // Some code...
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
}
```

Vamos a crear ahora a **EscritoraConRuta** como se indica en el esquema anterior, pero indicando que es una subclase de Escritora:

```
public class EscritoraConRuta extends Escritora{

    private String ruta = null;

    public void escribirTexto (String texto) {
        // Some code...
    }

    public String getRuta() {
        return ruta;
    }

    public void setRuta(String ruta) {
        this.ruta = ruta;
    }

}
```

Vamos a ve qué sucede si intentamos instanciar **Escritora** y **EscritoraConRuta**.

```
public static void main(String[] args) {

    Escritora escritora = new Escritora ();
    escritora.setNombre("nombre");
    escritora.setRuta("ruta");
    escritora.escribirTexto("some text");

    EscritoraConRuta escritoraConRuta = new EscritoraConRuta();
    escritoraConRuta.setNombre("nombre");
    escritoraConRuta.setRuta("ruta");
    escritoraConRuta.escribirTexto("some text");
}
```

Si nos fijamos, el código contiene un error evidente: no podemos usar método setRuta () de la variable **escritora** porque la clase **Escritora** no lo tiene definido.

Sin embargo, si te fijas, **escritoraConRuta** no genera un error al usar setNombre (). Esto se debe a que la clase de **EscritoraConRuta** lo hereda de **Escritora** y, por tanto, ella también lo tiene accesible.

Finalmente, ambas clases tienen un método que se llama escribirTexto (). En este caso, cada variable lanzará el método que se corresponde **con su clase** en concreto.