

PRIMER REPASO INGENIERIA DE SOFTWARE

JUAN SEBASTIAN RODRIGUEZ BLANCO

Universidad De Cundinamarca, Extensión Chía

Facultad De Ingeniería

Ingeniería de Sistemas y Computación Arquitectura de Computadores

William Matallana

Chía, Cundinamarca

24 de octubre del 2025

INTRODUCCION

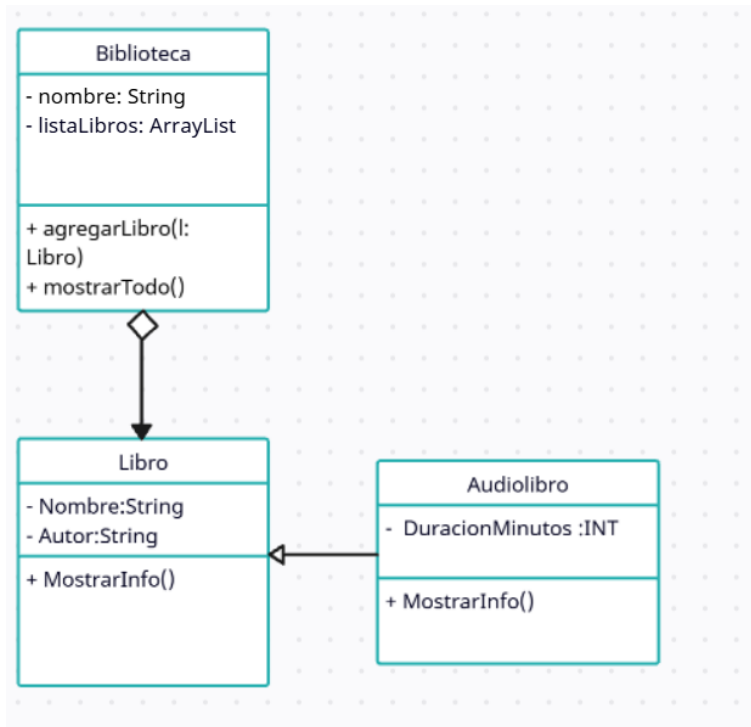
La Programación Orientada a Objetos (POO) es un paradigma diseñado para estructurar el software mediante clases, que actúan como plantillas, y objetos, que son las instancias reales de esas plantillas. Su potencia reside en la interacción de sus pilares: los atributos definen el estado, mientras que los métodos determinan el comportamiento; la herencia permite que nuevas clases hereden y extiendan funcionalidades de clases existentes para reutilizar código; el encapsulamiento protege la integridad de los datos mediante modificadores de acceso; y el polimorfismo otorga la flexibilidad de que una misma acción se ejecute de formas distintas según el objeto, logrando así sistemas escalables y fáciles de mantener.

EJERCICIO

Desarrollaremos un software diseñado para gestionar el inventario de una biblioteca de forma eficiente. El sistema se basa en una estructura de herencia donde definimos una clase padre llamada Libro, la cual centraliza los atributos comunes como títulos y autores. A partir de este molde, creamos la clase Audiolibro, que hereda automáticamente la información de nombre y autor, permitiéndonos añadir sus características propias (como duración) sin repetir código.

Además, el sistema integra una relación de agregación mediante la clase Biblioteca, la cual actúa como un contenedor dinámico que agrupa y gestiona todos los recursos. Esta organización nos permite aplicar los pilares de la POO: el encapsulamiento para proteger la información, el polimorfismo para manejar distintos formatos bajo una misma lista, y la abstracción para organizar el catálogo de forma profesional.

DIAGRAMA DE CLASES



CODIGO

clase Libro

```
1 package org.example;
2
3 public class Libro { 3 usages 1 inheritor
4     private String titulo; 4 usages
5     private String autor; 4 usages
6
7
8     public Libro(String titulo, String autor) { 2 usages
9         this.titulo = titulo;
10        this.autor = autor;
11    }
12
13
14    // --- GETTERS Y SETTERS ---
15    public String getTitulo() { return titulo; } no usages
16    public void setTitulo(String titulo) { this.titulo = titulo; } no usages
17
18    public String getAutor() { return autor; } no usages
19    public void setAutor(String autor) { this.autor = autor; } no usages
20
21
22
23    // --- TO STRING ---
24    @Override 1 override
25    public String toString() {
26        return "Titulo: " + titulo + ", Autor: " + autor;
27    }
28 }
```

Clase Audiolibro

```
1 package org.example;
2
3 public class Audiolibro extends Libro { 2 usages
4     private int duracionMinutos; 4 usages
5
6     public Audiolibro(String titulo, String autor, int duracionMinutos) { 1 usage
7         super(titulo, autor);
8         this.duracionMinutos = duracionMinutos;
9     }
10
11    // --- GETTER Y SETTER PROPIO ---
12    public int getDuracionMinutos() { return duracionMinutos; } no usages
13    public void setDuracionMinutos(int duracionMinutos) { this.duracionMinutos = duracionMinutos; } no use
14
15    // --- TO STRING (Sobrescrito) ---
16    @Override
17    public String toString() {
18        // Llamamos al toString de Libro y le pegamos el dato extra
19        return super.toString() + ", Duración: " + duracionMinutos + " min";
20    }
21 }
```

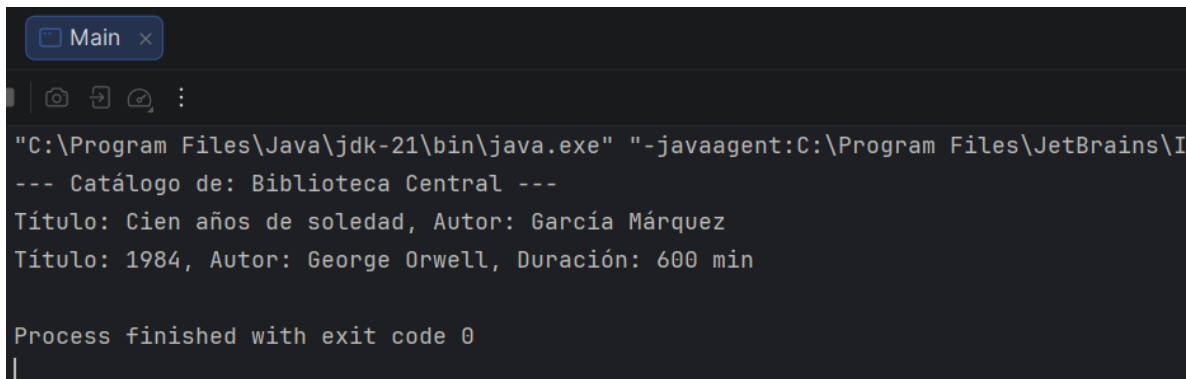
Clase Biblioteca

```
1 package org.example;
2
3 import java.util.ArrayList;
4
5 public class Biblioteca { 2 usages new *
6     private String nombre; 2 usages
7     // AGREGACIÓN: La biblioteca "tiene" una lista de libros
8     private ArrayList<Libro> listaLibros; 3 usages
9
10    public Biblioteca(String nombre) { 1 usage new *
11        this.nombre = nombre;
12        this.listaLibros = new ArrayList<>();
13    }
14
15    public void agregarRecurso(Libro recurso) { 2 usages new *
16        listaLibros.add(recurso);
17    }
18
19    public void mostrarCatalogo() { 1 usage new *
20        System.out.println("--- Catálogo de: " + nombre + " ---");
21        for (Libro l : listaLibros) {
22            System.out.println(l); // Polimorfismo en acción
23        }
24    }
25 }
```

Clase Main

```
1 package org.example;
2
3 public class Main {
4     public static void main(String[] args) {
5         // 1. Creamos la biblioteca
6         Biblioteca miBiblioteca = new Biblioteca(nombre: "Biblioteca Central");
7
8         // 2. Creamos los objetos (Instanciación)
9         Libro l1 = new Libro(titulo: "Cien años de soledad", autor: "García Márquez");
10        Audiolibro a1 = new Audiolibro(titulo: "1984", autor: "George Orwell", duracionMinutos: 600);
11
12        // 3. Los conectamos (Agregación)
13        miBiblioteca.agregarRecurso(l1);
14        miBiblioteca.agregarRecurso(a1);
15
16        // 4. Mostramos el resultado
17        miBiblioteca.mostrarCatalogo();
18    }
19 }
```

Muestra en consola

A screenshot of a Java IDE's console window. The window has a title bar with a tab labeled 'Main'. Below the title bar is a toolbar with icons for home, copy, paste, and a menu. The console output shows the execution of a Java program. The first line is the command: "C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\I...". The output consists of three lines: "--- Catálogo de: Biblioteca Central ---", "Título: Cien años de soledad, Autor: García Márquez", and "Título: 1984, Autor: George Orwell, Duración: 600 min". At the bottom, it says "Process finished with exit code 0".

```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\I
--- Catálogo de: Biblioteca Central ---
Título: Cien años de soledad, Autor: García Márquez
Título: 1984, Autor: George Orwell, Duración: 600 min

Process finished with exit code 0
```

CONCLUSIONES

Se evidencio lo sencillo que es trabajar cuando ya tienes un diseño definido ya que al usar clases como plantillas, puedes crear todos los libros que necesites rápidamente, sabiendo que todos van a funcionar igual de bien y tendrán la misma estructura.

También se evidencio el ahorro de tiempo con la herencia al aplicar la herencia fue clave para no trabajar doble permitió reciclar lo que ya había escrito en la clase principal (Libro) y usarlo en la otra, logrando que el código se vea mucho más limpio, ordenado y, sobre todo, fácil de entender.

LINK REPOSITORIO

<https://github.com/sjrodriguez/repaso-1/tree/main>

BIBLIOGRAFÍA

Moya, R. (8 de abril de 2014). *Herencia en Java, con ejemplos*. Jarroba.

<https://jarroba.com/herencia-en-la-programacion-orientada-a-objetos-ejemplo-en-java/>