

# eCatalog

---



## Sistema de Gestión de Biblioteca: ¡Tu Primera Aventura con Java de verdad!

¡Hola futuro desarrollador! 🙌 Te damos la bienvenida a tu proyecto de gestión de biblioteca.

No te preocupes, no vamos a construir la Biblioteca del Vaticano de golpe... iremos paso a paso, ¡como cuando aprendiste a montar en bici!



### ¿Qué vamos a construir?

Un sistema para gestionar libros que hará que hasta la señora de tu biblioteca local se ponga celosa. Lo haremos en varias fases, cada una más emocionante que la anterior (¡lo prometemos!).



### Las Fases del Proyecto (Total: 100 puntos)

#### Fase 1: Los Cimientos 🏗️ (20 puntos)

\*Donde todo empieza y rezamos para que compile\*

1. Implementación básica de Book (15 puntos):
  - Constructor con validación de null (3 puntos)
  - equals() y hashCode() correctos (5 puntos)
  - toString() bien formateado (2 puntos)
  - Implementación de IBook (5 puntos)
2. Pruebas Fase1.java pasan correctamente (5 puntos)

Lo que DE VERDAD hay que hacer:

1. Implementar la clase `Book` con:

- El constructor debe verificar que ningún parámetro sea null y lanzar `IllegalArgumentException` con el mensaje "Ningún campo puede ser null" si alguno lo es
- `equals()` debe comparar solo el ISBN de los libros, ignorando el resto de atributos
- `hashCode()` debe devolver el hash del ISBN para mantener

consistencia con `equals()`. ¡PISTA SÚPER SECRETA! 🤔

`return isbn.hashCode();` Si la fastidias con esto, te mandamos directamente a programar en COBOL con una

máquina de escribir... ¡y sin café! 💀

- `toString()` debe devolver una representación en String con todos los atributos del libro  
Ejemplo: `"Book{title='Don Quijote', author='Cervantes', isbn='123', publicationYear=1605}"`
- Implementar la interfaz `IBook` (¡no la toques, solo impléméntala!)

## 2. Pruebas en `Fase1.java`:

- Crear libros y verificar que sus datos son correctos
- Comprobar que dos libros con mismo ISBN son iguales
- Intentar crear libros con null y ver que explota (controladamente)

## Fase 2: El Contenedor Universal 🌌 (30 puntos)

*\*Donde los genéricos te harán dudar de tu existencia\**

La misión secreta (si decides aceptarla... aunque realmente no tienes opción 😈):

1. Crear `Catalog<T>` que implemente `ICatalog<T>`:
  - Necesitarás una estructura para almacenar elementos de tipo T (¿ArrayList tal vez? 🤔)
  - `addItem` debe devolver `true` si añade el elemento, `false` si ya existía
  - `removeItem` debe eliminar el elemento (y no explotar si no existe)

- `getAllItems` debe devolver una copia de la colección (¡no seas tacaño!)
- 2. Las pruebas verificarán:
  - Que funciona con Strings (fácil, como un paseo por el parque 🌳)
  - Que funciona con Books (aquí es donde el `equals` importa y donde empezarás a apreciar haberlo implementado bien)
  - Que no permite duplicados (¡tendrás que verificar antes de añadir! No queremos gemelos malvados en nuestro catálogo)
  - Que manejas las colecciones como un ninja 🥷 (sin efectos secundarios, sin referencias directas)

¡PISTA DE ORO! ✨ Recuerda que `T` puede ser CUALQUIER cosa. Sí, incluso ese tipo raro que tu compañero inventó ayer.







- Implementación de `Catalog<T>` (25 puntos):
  - Estructura de datos correcta (5 puntos)
  - `addItem` con control de duplicados (8 puntos)
  - `removeItem` funcionando (5 puntos)
  - `getAllItems` devolviendo copia (7 puntos)
- Pruebas `Fase2.java` pasan correctamente (5 puntos)


### Fase 3: La Biblioteca Toma Forma 🏛️ (25 puntos)

*\*Donde juntas las piezas y rezas a todos los dioses de la programación\** 🙏

El plan maestro (o cómo convertirte en el señor de los libros):

1. Implementar ``Library`` usando tu flamante ``Catalog<Book>`` (¡ese que te costó tres noches de insomnio!).
2. Debes implementar estos métodos explosivos (literalmente):


- ``addBook``: Si el ISBN ya existe → ¡BOOM!   
(DuplicateBookException)  
¡Como cuando intentas registrarte en una web y te dicen  
que tu email ya existe! 
- ``removeBook``: Si el ISBN no existe → ¡BOOM!   
(BookNotFoundException)  
Es como buscar las llaves que JURASTE que dejaste en la  
mesa... ¡pero no están! 
- ``findByIsbn``: Si no encuentra el libro → ¡BOOM!   
(BookNotFoundException)  
Similar a cuando buscas ese calcetín perdido... ¡sabes  
que existe pero no aparece! 

¡MEGA PISTA!  Si tu código explota más que una fábrica de  
fuegos artificiales, vas por buen camino.  
¡Pero que explote con ESTILO y las excepciones correctas!

1. Implementación de Library (20 puntos):
  - `addBook` con gestión de duplicados (7 puntos)
  - `removeBook` con validación (7 puntos)
  - `findByIsbn` con excepciones (6 puntos)
2. Pruebas Fase3.java pasan correctamente (5 puntos)

## Fase 4: Los Superpoderes (25 puntos)

*\*Donde tu biblioteca se convierte en una base de datos\**

La batalla final (¡prepara tu capa de superhéroe!  ):

1. Añadir búsquedas avanzadas a `Library` (porque buscar solo por ISBN es muy 1999):

- `findByAuthor`: Encuentra todos los libros de un autor  
Como buscar todas las fotos de tu ex en Facebook... ¡pero

con libros! 

- `findByYear`: Encuentra todos los libros publicados en un año específico

Para cuando quieres saber qué se publicó el año que


nacistes (¡y deprimirte!) 


- `getAllBooks`: Devuelve TODOS los libros  
Como pedir todo el menú en el restaurante... ¡pero sin


remordimientos! 



2. Reglas de oro (más sagradas que el manual de instrucciones de IKEA):

- Si no hay resultados → ArrayList vacío (¡como tu nevera a

fin de mes! )

- Siempre devolver copias (porque compartir referencias es como prestar tu cepillo de dientes... ¡NO!) 

- Validar parámetros (si recibes null, responde como un profesor de matemáticas: "¡Esto no es aceptable!" )

- Las búsquedas deben ser tan rápidas que Flash se ponga celoso  

• Búsquedas avanzadas (20 puntos):

- `findByAuthor` eficiente (7 puntos)
- `findByYear` correcto (6 puntos)
- `getAllBooks` bien implementado (4 puntos)
- Manejo de casos especiales (3 puntos)

• Pruebas `Fase4.java` pasan correctamente (5 puntos)



## Cómo Probar Tu Código

Cada fase tiene su propia clase de prueba:

- ``Fase1.java``: Para ver si tus libros son libros
- ``Fase2.java``: Para jugar con tu catálogo mágico
- ``Fase3.java``: Para probar tu biblioteca básica
- ``Fase4.java``: Para presumir de biblioteca futurista



## Requisitos Importantes (¡No los ignores!)

Para la clase Book:

- El ISBN es SAGRADO (es el DNI del libro)
- Si dos libros tienen el mismo ISBN, son el mismo libro (aunque uno esté en español y otro en klingon)
- ¡NO a los campos vacíos! (null = 🦴)

Para el Catalog:


- No le gustan los duplicados (es único y especial)
- Usa ``equals()`` para ver si algo está repetido (como cuando buscas a tu gemelo)
- Cuando devuelve la lista de items, da una copia (¡no seas tacaño!)

Para Library:

- Maneja los errores con elegancia (nada de pánico)
- Busca rápido (no queremos que el usuario se haga viejo esperando)
- No compartas tus listas internas (son privadas, ¡respeta!)




## Notas Finales (¡Lee esto o llora después!)

- No toques las interfaces (están ahí por algo)
- Comenta tu código (tu yo del futuro te lo agradecerá)
- Maneja los null como si fueran 
- Si algo puede fallar, ¡que falle con estilo! (excepciones bien manejadas)
- Las búsquedas que no encuentran nada devuelven listas vacías (¡no null!)



### ¿Perdido?


¡No pasa nada! Roma no se construyó en un día, y tu biblioteca tampoco tiene que estar perfecta a la primera.


Si te atascas, respira hondo, maldice a tu profesor  y revisa el código paso a paso.



### ¡A Programar!

Recuerda: el mejor código es el que funciona. Y el segundo mejor es el que funciona Y se entiende.

¡Buena suerte! 

P.D.: Si encuentras un bug, no es un bug, ¡es una característica especial! (bueno, no... arrégalo )



## Consejos para no morir en el intento

### 1. Orden de implementación sugerido:

- Primero ``Book`` (es la base de todo)
- Luego ``Catalog`` (pruébalo con Strings primero)
- Después ``Library`` básica (add/remove/findByIsbn)
- Finalmente las búsquedas avanzadas

### 2. Cuando las pruebas fallen (que fallarán):

- Lee el error (sí, LÉELO de verdad)
- Usa el debugger (es tu amigo, no tu enemigo)
- Revisa los requisitos (están por algo)



## Obtención de Puntuaciones

### Fase 1 (20 puntos)

Cuando ejecutas ``Fase1.java``:

- ☒ Prueba 1 - Getters (5 puntos):
  - Todos los getters devuelven valores correctos
  - Constructor valida nulls correctamente
- ☒ Prueba 2 - equals/hashCode (10 puntos):
  - equals compara solo ISBN (5 puntos)
  - hashCode consistente con equals (5 puntos)
- ☒ Prueba 3 - toString (5 puntos):
  - Incluye todos los campos
  - Formato correcto

### Fase 2 (30 puntos)

Cuando ejecutas ``Fase2.java``:

- ☒ Prueba 1 - Strings (10 puntos):
  - Añadir elementos únicos (3 puntos)
  - Control de duplicados (4 puntos)



- Eliminación correcta (3 puntos)
- ☒ Prueba 2 - Eliminación (10 puntos):
  - Eliminar existente (5 puntos)
  - Manejo de no existentes (5 puntos)
- ☒ Prueba 3 - Books (10 puntos):
  - Control de duplicados por ISBN (5 puntos)
  - Gestión de colección (5 puntos)

### Fase 3 (25 puntos)



Cuando ejecutas `Fase3.java`:

- ☒ Prueba 1 - Añadir libros (8 puntos):
  - Añadir correctamente (4 puntos)
  - Verificar colección (4 puntos)
- ☒ Prueba 2 - Eliminar libro (7 puntos):
  - Eliminar existente (4 puntos)
  - Verificar estado (3 puntos)
- ☒ Prueba 3 y 4 - Excepciones (10 puntos):
  - DuplicateBookException (5 puntos)
  - BookNotFoundException (5 puntos)

### Fase 4 (25 puntos)

Cuando ejecutas `Fase4.java`:

- ☒ Prueba 1 y 2 - Búsqueda ISBN (7 puntos):
  - Encontrar existente (4 puntos)
  - Manejar no existente (3 puntos)
- ☒ Prueba 3 y 4 - Búsqueda autor (8 puntos):
  - Encontrar libros de autor (4 puntos)
  - Manejar autor inexistente (4 puntos)
- ☒ Prueba 5 - Consistencia (10 puntos):
  - Mantener coherencia tras eliminación (5 puntos)
  - Estado correcto del catálogo (5 puntos)

💡 NOTA: Cada  en las pruebas significa que esa parte está correcta. Si ves , revisa esa sección específica.