

---

# PROYECTO BIBLIOTECARIO MONGODB Y JAVA

---

TRABAJO DE AMPLIACIÓN

ISMAEL ÁNGEL SORIA RIVERO  
BASES DE DATOS Y PROGRAMACIÓN  
1º DAM

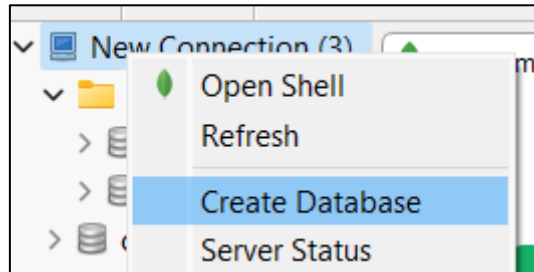
# ÍNDICE

1.- CREAR BASE DE DATOS EN MONGODB	-----	2-5
2.- CONFIGURACIÓN EN ECLIPSE	-----	6-8
3.- CONEXIÓN CON MONGODB	-----	9-12
• Configuración archivo pom.xml	-----	9
• Configuración de nuestro main	-----	10-12
4.- CONECTARNOS A UNA BD YA CREADA	-----	13
5.- ADICIÓN DE DATOS	-----	14-17
6- PROYECTO: “BIBLIOTECARIO”	-----	18-29
• ANEXO FUNCIONES	-----	28
7.- BIBLIOGRAFÍA	-----	30

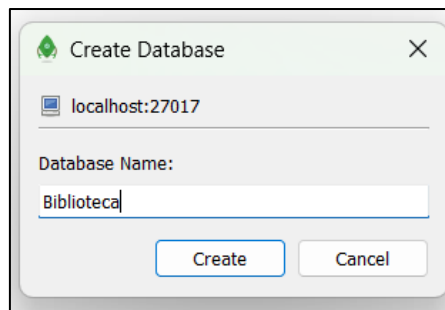
## 1.- CREAR BASE DE DATOS EN MONGODB

Entramos en Robo 3T, y empezamos a construir nuestra base de datos, se denominará biblioteca y tendrá tres colecciones.

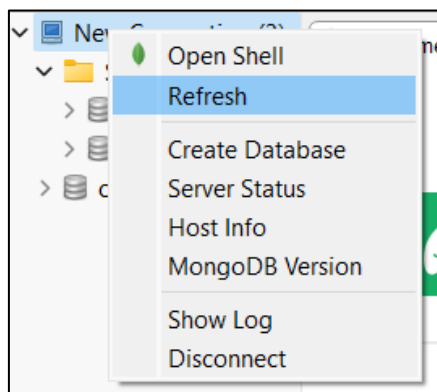
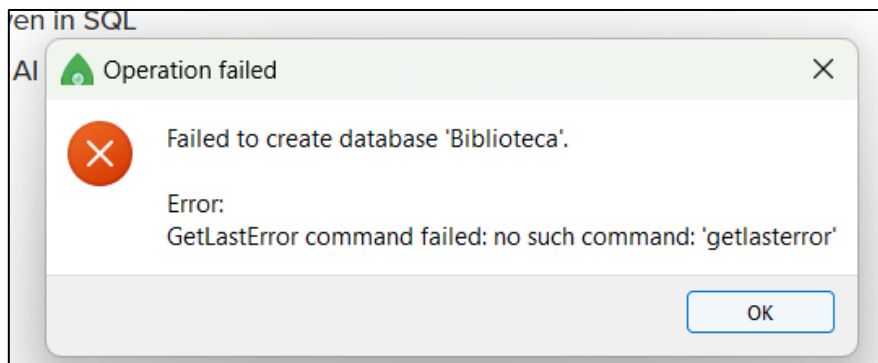
Boton derecho encima de New Connection, créate Database.



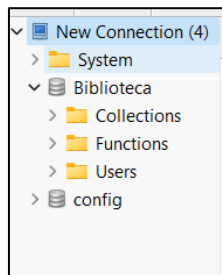
Insertamos Biblioteca, podemos fiarnos que se nos muestra el puerto 27017.



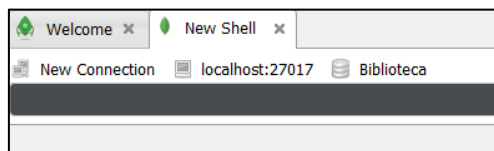
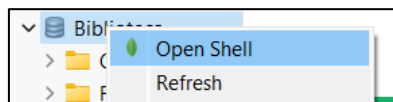
Se muestra un mensaje de erro, pero no debemos preocuparnos. Botón derecho encima de New Connection y seleccionamos Refresh.



Ya nos aparece nuestra Base de Datos Biblioteca.

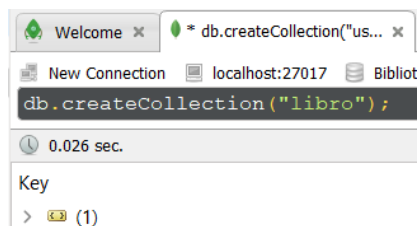


Creamos las colecciones. Para ello clicamos botón derecho encima de Biblioteca, y seleccionamos Open Shell. Se nos abre la terminal. Podemos nuestros libros.



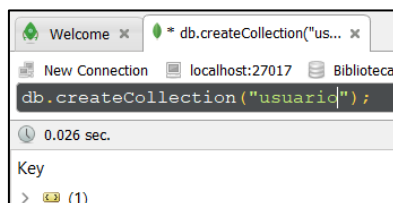
## COLECCIÓN LIBROS

Campos => título, autor, prestado.



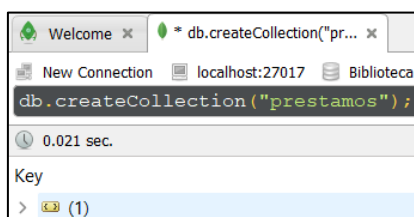
## COLECCIÓN USUARIO

Campos => nombre, email, librosLeidos(Historial de los libros leidos).



## COLECCIÓN PRESTAMOS

Campos => idTitulo,Título, idUsuario. Usuario, FechaPrestamo, FechaDevolución,estado



## INSERTAR EN LA BASE DE DATOS LIBROS Y USUARIOS.

El esquema será el mismo usando:

Insertar Varios

### Insertar uno

```
db.ColeccionElegida.insertOne({  
    campo : valor,  
    campo:valor  
})
```

```
Db.ColeccionElegida.insertMany([  
    {  
        campo : valor,  
        campo:valor  
    },  
    {  
        campo : valor,  
        campo:valor  
    }  
])
```

Insertaremos 20 libros generados por ChatGPT y dos usuarios esos los introduciré manualmente.

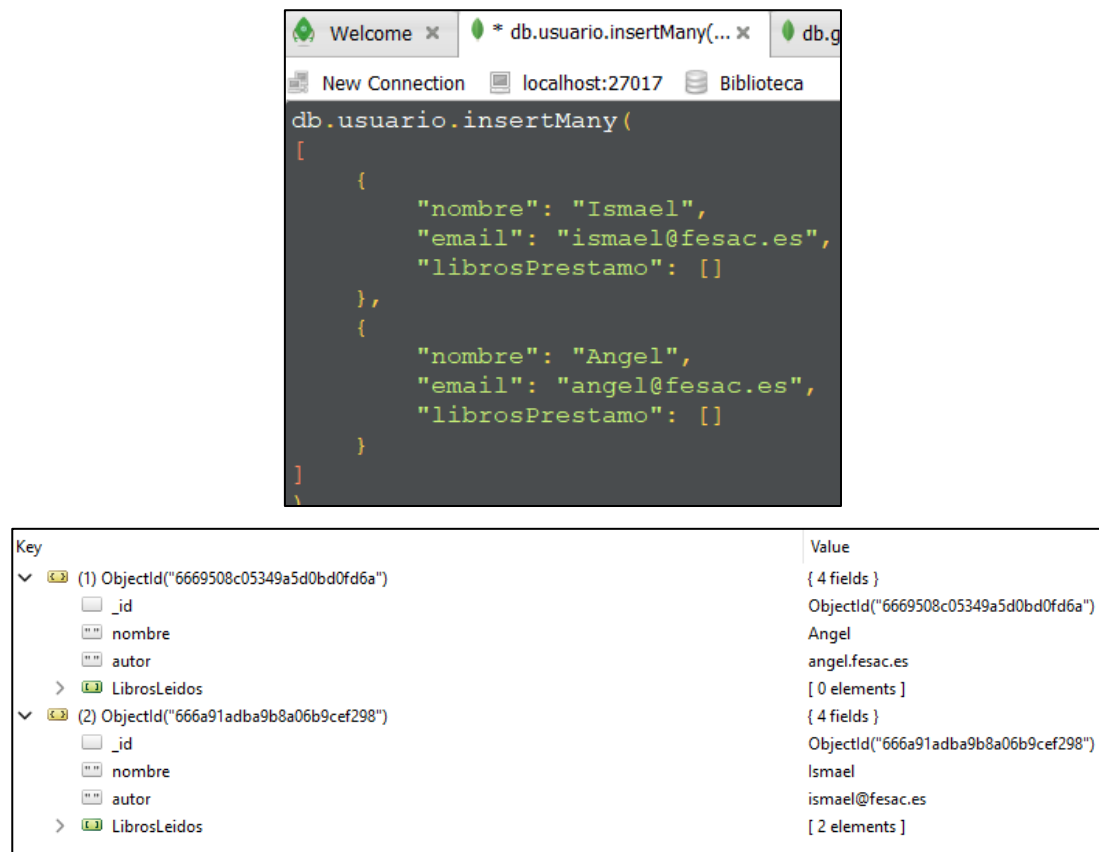


```
db.libro.insertMany(  
  {  
    "titulo": "Don Quijote de la Mancha",  
    "autor": "Miguel de Cervantes",  
    "prestado": false  
  },  
  {  
    "titulo": "Moby Dick",  
    "autor": "Herman Melville",  
    "prestado": false  
  },  
  {  
    "titulo": "Orgullo y prejuicio",  
    "autor": "Jane Austen",  
    "prestado": false  
  },  
)
```



Key	Value
(1) ObjectId("6667262c2f6ad68ecb35f84")	{ 4 fields }
_id	ObjectId("6667262c2f6ad68ecb35f84")
titulo	Don Quijote de la Mancha
autor	Miguel de Cervantes
prestado	false

Insertamos los dos usuarios manualmente



The screenshot shows a MongoDB IDE window with the following content:

```
db.usuario.insertMany([
  {
    "nombre": "Ismael",
    "email": "ismael@fesac.es",
    "librosPrestamo": []
  },
  {
    "nombre": "Angel",
    "email": "angel@fesac.es",
    "librosPrestamo": []
  }
])
```

Below the code editor, a table displays the results of the operation:

Key	Value
✓ (1) ObjectId("6669508c05349a5d0bd0fd6a")	{ 4 fields }
_id	ObjectId("6669508c05349a5d0bd0fd6a")
nombre	Angel
autor	angel.fesac.es
LibrosLeidos	[ 0 elements ]
✓ (2) ObjectId("666a91adba9b8a06b9cef298")	{ 4 fields }
_id	ObjectId("666a91adba9b8a06b9cef298")
nombre	Ismael
autor	ismael@fesac.es
LibrosLeidos	[ 2 elements ]

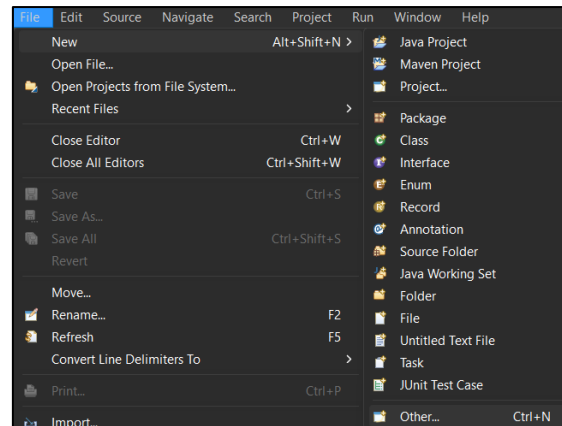
Después investigando, no es del todo imprescindible crear nuestra base de datos lo primero, ya que desde Java una vez conectado a Mongo, si no se encuentra la colección o algún campo en concreto se genera automáticamente. Pero si debemos comentar que a efectos de tener el proyecto más organizado y comprender su estructura, es desde mi punto de vista, una mejor opción, crear o al menos tener clara la estructura de la base de datos y partiendo de ahí ir realizando nuestro programa.

Ahora tendremos que configurar Eclipse para ello vamos a crear un proyecto Maven. Este tipo de proyecto incluye ya funcionalidades y carpetas concretas que nos facilitarán la conexión con la base de datos.

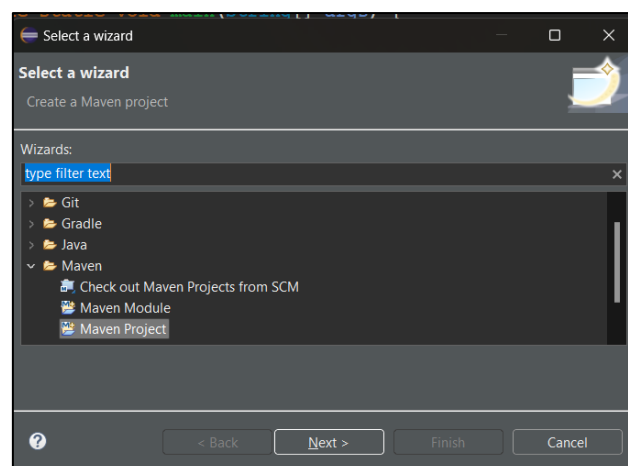
## 2.- CONFIGURACIÓN EN ECLIPSE

Crear un proyecto Maven.

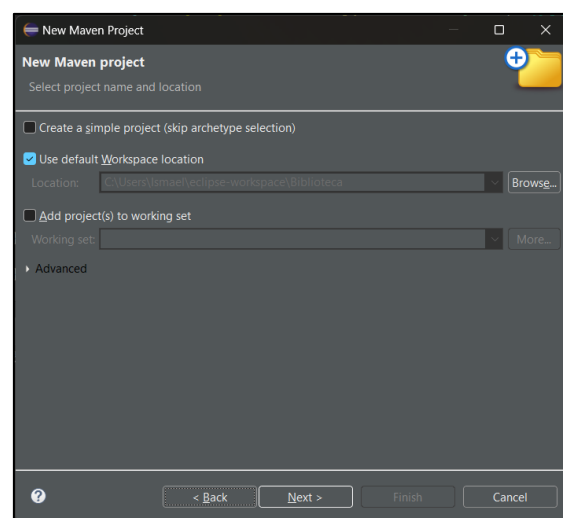
Seleccionamos File, New, Other.



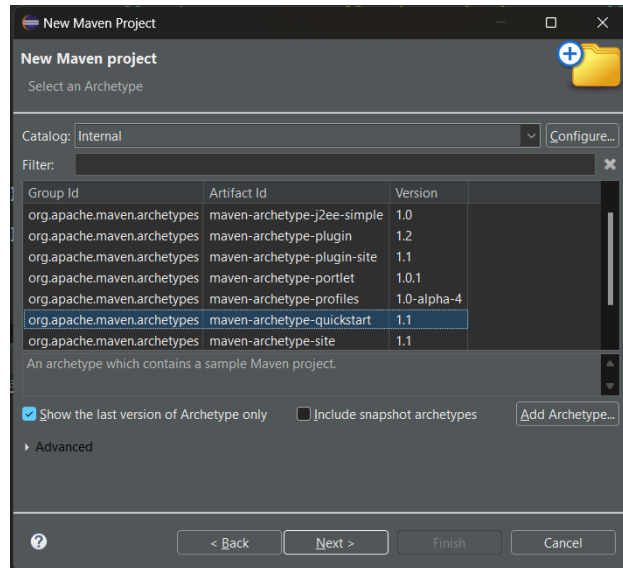
Se nos abre esta ventana en ella, seleccionamos dentro de la carpeta Maven, clicamos en Maven Project.



Indicamos donde vamos a alojar nuestro Maven Project. Elegimos por Defecto.



Tenemos que seleccionar el arquetipo. El arquetipo es una plantilla predefinida que se utiliza para crear Proyectos, proporcionando una estructura inicial con archivos y configuraciones básicas. Seleccionamos el arquetipo “Maven-archetype-quickstart”, es uno de los mas simples, para un proyecto inicial como el nuestro es una buena elección.



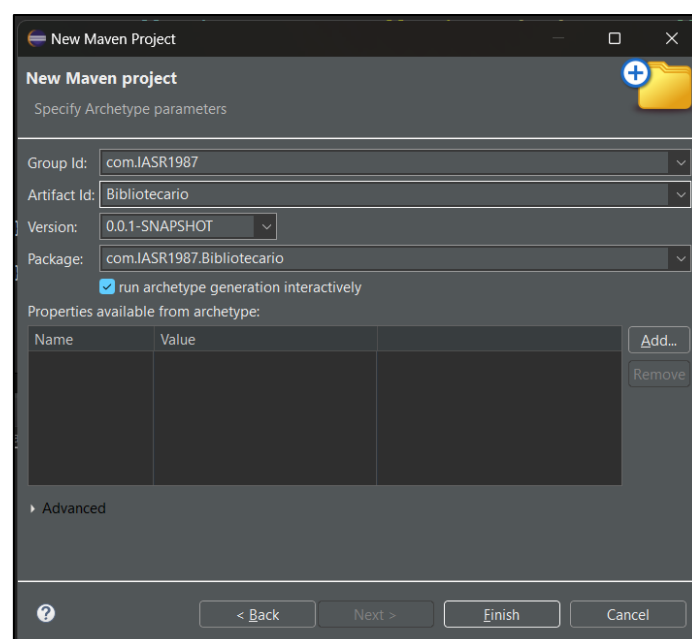
Tenemos que configurar nuestro Proyecto Maven introduciendo:

Group ID:

- Es un identificador, ormalmente es el domino dado la vuelta.
- fesac.com => com.fesac
- Si es local => com.NombreUsuario

Artifact Id:

- Nombre del proyecto o artefacto, debe ser único

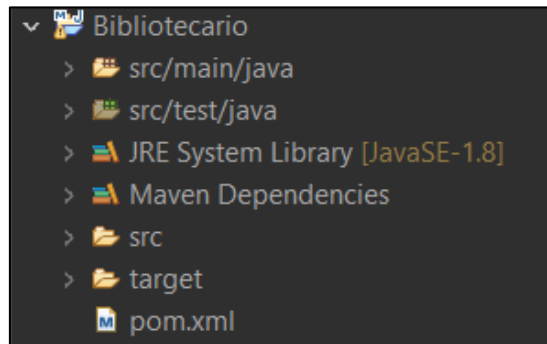




Se inicia un proceso de descarga en nuestra consola

```
[INFO] -----
[INFO] Using following parameters for creating project from Old (1.x) Archetype: maven-archetype-quickstart:1.1
[INFO] -----
[INFO] Parameter: basedir, Value: C:\Users\Ismael\eclipse-workspace
[INFO] Parameter: package, Value: com.IASR1987.Bibliotecario
[INFO] Parameter: groupId, Value: com.IASR1987
[INFO] Parameter: artifactId, Value: Bibliotecario
[INFO] Parameter: packageName, Value: com.IASR1987.Bibliotecario
[INFO] Parameter: version, Value: 0.0.1-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: C:\Users\Ismael\eclipse-workspace\Bibliotecario
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 13:07 min
[INFO] Finished at: 2024-06-11T09:15:31+02:00
[INFO] -----
```

Observamos que se nos ha aparecido una carpeta denominada Bibliotecario en nuestro Explorador, como ya dijimos se nos generaría una estructura para nuestro proyecto, esta responderá al arquetipo elegido.



Es de importancia el archivo pom.xml. este archivo XML describe la configuración del proyecto Maven, incluye dependencias, la estructura, los plugins a utilizar y cualquier otro tipo de información importante para nuestro proyecto. Se recoge en él, toda la información para construir, gestionar y compilar el proyecto.

```
App.java  Bibliotecario/pom.xml x
http://maven.apache.org/xsd/maven-4.0.0.xsd (xsi:schemaLocation with catalog)
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5   <groupId>com.IASR1987</groupId>
6   <artifactId>Bibliotecario</artifactId>
7   <version>0.0.1-SNAPSHOT</version>
8   <packaging>jar</packaging>
9   <name>Bibliotecario</name>
10  <url>http://maven.apache.org</url>
11
12  <properties>
13    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
14  </properties>
15
16  <dependencies>
17    <dependency>
18      <groupId>junit</groupId>
19      <artifactId>junit</artifactId>
20      <version>3.8.1</version>
21      <scope>test</scope>
22    </dependency>
23  </dependencies>
24 </project>
25
26
```

### 3.- CONEXIÓN CON MONGO DB

Una vez ya realizado nuestro Maven Project, tenemos que configurar la conexión a nuestra base de datos en MongoDB.


#### *CONFIGURACIÓN ARCHIVO POM.XML*

Primero debemos configurar nuestro archivo pom.xml, debemos añadirles las dependencias del controlador MongoDB Java, nos facilitará las herramientas necesarias para poder interactuar nuestra base de datos con nuestra aplicación Java.

Tenemos que tener en cuenta que ya existe una pestaña dependencias, por lo que debemos agregar esta a continuación de la que se encontraba ya.

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.mongodb</groupId>
    <artifactId>mongodb-driver-sync</artifactId>
    <version>5.1.0</version>
  </dependency>
</dependencies>
```

Después debemos añadir las dependencias SLF4J y Logback. Estas dependencias nos permiten registrar mensajes en tu aplicación referentes a advertencias, errores, u otra información relevante que se producen durante la ejecución del programa. Además, podemos configurar lo que queremos hacer con esos mensajes. Es importante si tenemos que depurar nuestra aplicación y obtener un registro de lo que esta sucediendo.

```
!-- añadimos esta dependencia para tener un regsitro de mensajes -->
<dependency>
  <!-- SLF4J API -->
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-api</artifactId>
  <version>1.7.32</version>
</dependency>
<!-- Implementación de Logback para SLF4J -->
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>1.2.6</version>
</dependency>
```

Ya se encuentra configurada nuestro archivo pom.xml con las dependencias necesarias.

## CONFIGURAR CONEXIÓN EN NUESTRO MAIN

En la carpeta src/main/java se creo, por defecto una clase Java de nombre App.java, la utilizaremos como Main de nuestra aplicación. Si la abrimos comprobamos que dentro existe un hola Mundo. Borramos el contenido.

```
App.java ×
1 package com.IASR1987.Bibliotecario;
2
3 /**
4  * Hello world!
5  *
6  */
7 public class App
8 {
9     public static void main( String[] args )
10    {
11        System.out.println( "Hello World!" );
12    }
13 }
14
```

Comenzaremos configurando nuestra conexión para ello utilizaremos dos clases del controlador MongoDB que hemos instalado:

*ConnectionString* => crea una cadena de conexión a MongoDB. Para construirla seguimos este esquema =>

- “mongodb://host1:NumeroDeNuestroPuerto
- “mongodb://host1:27017”
- Varios Puertos => “mongodb://host1:27017, host2:271018”

```
ConnectionString newConexion = new ConnectionString("mongodb://host1:27017");
```

Nos salta un error, ya que debemos importar ConnectionString

```
ConnectionString("mongodb://host1:27017");
// Import 'ConnectionString' (com.mongodb)
```

Nos debería quedar algo así:

```
package com.IASR1987.Bibliotecario;

import com.mongodb.ConnectionString;

public class App
{
    public static void main( String[] args )
    {
        ConnectionString newConexion = new ConnectionString("mongodb://host1:27017");
    }
}
```

*MongoClient* => crea una instancia para representar la conexión al servidor de MongoDB. Se utiliza el método estático `create()` disponible en *MongoClient*, para crear una instancia utilizando la cadena de conexión creada anteriormente,

```
MongoClient Ismael = MongoClient.create(newConexion);
```

Vemos que como en la clase anterior da un error, debemos importar la clase *MongoClient* y *MongoClients*

```
MongoClient Ismael = MongoCl
```

```
    - Import 'MongoClient' (com.mongodb.client)
```

```
MongoClients.create(newConexion);
```

```
    - Import 'MongoClients' (com.mongodb.client)
```

Nos quedaría algo así:

```
package com.IASR1987.Bibliotecario;

import com.mongodb.ConnectionString;
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoClients;

public class App
{
    public static void main( String[] args )
    {
        ConnectionString newConexion = new ConnectionString("mongodb://host1:27017");
        MongoClient Ismael = MongoClient.create(newConexion);
    }
}
```

Si le damos a Ejecutar se nos genera el siguiente mensaje, la conexión se está produciendo de manera correcta.

```
<terminated> App [Java Application] C:\Users\ismael\AppData\Local\Temp\org.eclipse.justopenjdk.hotspot.jre.full.win32.x86_64_17.0.11.v20240426-1830\jre\bin\javaw.exe (11 jun 2024 10:30:49 - 10:30:50) [pid:1192]
10:30:50.180 [main] DEBUG org.mongodb.driver.connection - Connection pool created for host1:27017 using options maxIdleTimeMS=0, minPoolSize=0, maxPoolSize=100, maxConnecting=2, waitQueue
10:30:50.190 [main] DEBUG org.mongodb.driver.cluster - Updating cluster description to {type=UNKNOWN, servers=[{address=host1:27017, type=UNKNOWN, state=CONNECTING}]
10:30:50.205 [main] INFO org.mongodb.driver.client - MongoClient with metadata {"driver": {"name": "mongo-java-driver|sync", "version": "5.1.0"}, "os": {"type": "Windows", "name": "Windo
```

Una vez ya tenemos configurada nuestra conexión debemos cerrar la conexión para ello:

```
//cerramos nuestra conexión
Ismael.close();
```

Por buenas prácticas, es aconsejable poner un try catch por si existiera algún problema en la conexión.

```
try {
    //creamos la cadena de conexión
    ConnectionString newConexion = new ConnectionString("mongodb://host1:27017");

    //nos conectamos a MongoDB
    MongoClient Ismael = MongoClients.create(newConexion);
    |
    //cerramos nuestra conexión
    Ismael.close();
} catch (Exception e) {
    //mensaje de error
    System.out.println("Error al establecer la conexión");
    //nos indicará el error
    e.printStackTrace();
}
```

## 4.- CONECTARNOS A UNA BASE DE DATOS YA CREADA EN MONGODB

Estamos conectados a MongoDB, pero ¿cómo accedemos a nuestra Base de Datos que ya creamos en MongoDB?. Tenemos que tener en cuenta que es indispensable crear primero la Database. He visto dos modos de realizar esto:

### 1.- Modificando ConnectionString:

Añadiéndole a continuación del puerto utilizado después de una barra inclinada el nombre de nuestra base de datos.

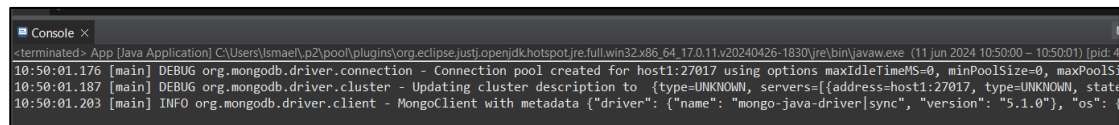
```
//creamos la cadena de conexión
ConnectionString newConexion = new ConnectionString("mongodb://host1:27017/Biblioteca");
```

### 2.- usando MongoDBase

Sin modificar ConnectionString, insertamos este comando. Teniendo en cuenta el nombre de nuestro MongoClient.

```
//nos conectamos a la base de datos
MongoDatabase database = Ismael.getDatabase("Biblioteca");
```

Si ejecutamos nos sale el mismo mensaje que pusimos anteriormente, así que la conexión es exitosa.



```
Console X
<terminated> App [Java Application] C:\Users\ismael.p2\poo\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.17.0.11.v20240426-1830\jre\bin\javaw.exe (11 Jun 2024 10:50:00 - 10:50:01) [pid: 4]
10:50:01.176 [main] DEBUG org.mongodb.driver.connection - Connection pool created for host1:27017 using options maxIdleTimeMS=0, minPoolSize=0, maxPoolSize=10
10:50:01.187 [main] DEBUG org.mongodb.driver.cluster - Updating cluster description to {type=UNKNOWN, servers=[{address=host1:27017, type=UNKNOWN, state=CONNECTING}]}
10:50:01.203 [main] INFO org.mongodb.driver.client - MongoClient with metadata {"driver": {"name": "mongo-java-driver|sync", "version": "5.1.0"}, "os": {}}
```

Ahora podemos modificar nuestra Database.

## 5.- ADICIÓN DE DATOS

Una vez tenemos configurada nuestra conexión queremos añadir datos, explicaremos el proceso. El primer paso es acceder a la colección en la que queremos actuar. Para ello debemos usar el siguiente comando:

```
//accedemos a la coleccion libro
MongoCollection<Document>Catalogo = Biblioteca.getCollection("libro");
```

Los errores son que debemos importar elementos.

```
MongoCollection<Document> Cata<Document> Catalogo = da
~ Import 'MongoCollection' (com.mongodb.client) ~ Import 'Document' (javax.swing.text)
```

- Nos fijamos en el uso del genérico Document, esto indica que la colección almacena los documentos en formato BSON, que el que soporta MongoDB.
- Un aspecto a tener en cuenta es que si la colección no existiera, MongoDb la crearía.
- Una vez dentro de la colección podemos modificarla como queramos

Quiero decir que hasta este momento Java y la base de Datos se conectaban correctamente, pero a partir de empezar a modificarla, me daba errores en la conexión teniendo que modificar la Connection-string:

```
//creamos la cadena de conexión
ConnectionString newConexion = new ConnectionString("mongodb://localhost:27017");
```

### *LISTAR LOS DOCUMENTOS DE LA COLECCIÓN*

Utilizamos FindIterable, que es un tipo de referencia definido por el driver de MongoDB para Java. Nos devuelve documento a documento todo el contenido de una colección. Después queremos imprimirlos todos por pantalla, para ello utilizamos un bucle for each. Tendremos en cuenta que todos los documentos viene en formato Document, debemos convertirlos mediante el método toJson(), para que pueda ser legible.

```
//listar todos los elementos de una lista
FindIterable<Document> libros = Catalogo.find();

//mostrar por pantalla
for (Document libro : libros) {
    System.out.println(libro.toJson());
}
```

## AÑADIR DOCUMENTOS A NUESTRA COLECCIÓN

Tenemos dos opciones:

### 1.- Añadir un solo elemento

Mediante la clase `InsertOneResult` se puede agregar un nuevo elemento a nuestra lista. Después de crear el objeto seleccionamos el conjunto de documentos y seleccionamos `insertOne`, debemos crear un nuevo `Document` con los campos que tenga el documento en la base de datos. Para ello usamos `.append` y entre paréntesis el nombre del campo, seguido de su valor.

```
//insertar un documento
InsertOneResult nuevoLibro = Catalogo.insertOne(new Document()
    .append("titulo", "El camino")
    .append("autor", "Miguel Delibes")
    .append("prestado", "false")
);
```

Entramos en Robo 3T y refrescamos nuestra base de datos y comprobamos que se ha agregado correctamente:

(21) ObjectId("666829404923dd03d3def88c")	{ 4 fields }	Object
id	ObjectId("666829404923dd03d3def88c")	ObjectId
titulo	El camino	String
autor	Miguel Delibes	String
prestado	false	String

### 2.- Añadir varios elementos

En este caso debemos seguir dos pasos primero crear una `List` de `Document` y después agregarles los libros. Esta lista después será pasada a la base de Datos.

```
//creamos la lista de Documentos
ArrayList<Document> nuevosLibros = new ArrayList<Document>();
```

Creamos los objetos:

```
//creamos los nuevos objetos y los introducimos en el arrayList
Document libro1 = new Document("titulo", "Viaje al Centro de la Tierra")
    .append("autor", "Julio Verne")
    .append("prestado", "false");

Document libro2 = new Document("titulo", "Los pilares de la Tierra")
    .append("autor", "Ken Follett")
    .append("prestado", "false");

Document libro3 = new Document("titulo", "La Celestina")
    .append("autor", "Fernando de Rojas")
    .append("prestado", "false");
```



Los añadimos a nuestro ArrayList

```
//agregamos al arrays
nuevosLibros.add(libro1);
nuevosLibros.add(libro2);
nuevosLibros.add(libro3);
```

Pasamos a insertarlos. Usamos la clase InsertManyResult, creamos el objeto haciendo un insertMany en nuestro Catalogo, pasándole por parámetro el arrayList que creamos antes.

```
//los insertamos
InsertManyResult AnadirLibros = Catalogo.insertMany(nuevosLibros);
```

Entramos de nuevo en Robo 3T, refrescamos y comprobamos que se hayan insertado nuestros libros.

▼ (22) ObjectId("66682dad232b506a3db7b7e1")	{ 4 fields }
_id	ObjectId("66682dad232b506a3db7b7e1")
titulo	Viaje al Centro de la Tierra
autor	Julio Verne
prestado	false
▼ (23) ObjectId("66682dad232b506a3db7b7e2")	{ 4 fields }
_id	ObjectId("66682dad232b506a3db7b7e2")
titulo	Los pilares de la Tierra
autor	Ken Follett
prestado	false
▼ (24) ObjectId("66682dad232b506a3db7b7e3")	{ 4 fields }
_id	ObjectId("66682dad232b506a3db7b7e3")
titulo	La Celestina
autor	Fernando de Rojas
prestado	false

### ACTUALIZAR UN DOCUMENTO

Si queremos actualizar un documento, en nuestro caso es necesario para cuando un libro sea prestado, tenemos que realizar dos procesos, uno de selección del documento a modificar y otro del campo a modificar.

Utilizamos la función updateOne, que primero usa un filtro para determinar el Documento a actualizar y en el otro se selecciona el campo a modificar.

```
//modificar un atributo de un documento
Catalogo.updateOne(Filters.eq("titulo", "Viaje al Centro de la Tierra"),
new Document("$set", new Document("prestado", "true")));
```

Comprobamos que prestado ha pasado a true.

▼ (22) ObjectId("66682dad232b506a3db7b7e1")	{ 4 fields }
_id	ObjectId("66682dad232b506a3db7b7e1")
titulo	Viaje al Centro de la Tierra
autor	Julio Verne
prestado	true

## ELIMINAR UN DOCUMENTO DE LA COLECCIÓN

Es parecido a actualizar al elemento, pero solo tenemos que filtrar el documento a eliminar.

```
//eliminar un elemento
Catalogo.deleteOne(Filters.eq("titulo", "Viaje al Centro de la Tierra"));
```

Entramos en Robo 3T, actualizamos y comprobamos que en la posición 22, que era en la que se encontraba Viaje al Centro de la Tierra, ahora aparece otro libro.

(22) ObjectId("66682dad232b506a3db7b7e2")	{ 4 fields }
_id	ObjectId("66682dad232b506a3db7b7e2")
titulo	Los pilares de la Tierra
autor	Ken Follett
prestado	false

## PROYECTO BIBLIOTECARIO

He desarrollado una biblioteca usando Java y MongoDB.

Comenzamos con la creación de la base de Datos en Mongo, esos pasos vienen reflejados en el punto 1 de este documento.

### *CONEXIÓN A MONGO*

Debemos tal como hemos visto en pasos anteriores conectarnos a MongoDB. He utilizado un try catch, para contener el error si la conexión es errónea.

```
//creamos la cadena de conexión
ConnectionString newConexion = new ConnectionString("mongodb://localhost:27017");

//nos conectamos a MongoDB, creando un cliente de conexión
MongoClient Ismael = MongoClient.create(newConexion);

//nos conectamos a la base de datos
MongoDatabase Biblioteca = Ismael.getDatabase("Biblioteca");

//accedemos a las colecciones
//accedemos a la coleccion libro
MongoCollection<Document>Catalogo = Biblioteca.getCollection("libro");
MongoCollection<Document>Usuarios = Biblioteca.getCollection("usuario");
MongoCollection<Document>Prestamos = Biblioteca.getCollection("prestamos");

//creamos Listas de los documentos para utilizarlos después
//listar todos los elementos de una lista
FindIterable<Document> catalogoLibros = Catalogo.find();
FindIterable<Document> listaUsuarios = Usuarios.find();
FindIterable<Document> registroPrestamos = Prestamos.find();
```

Observamos que listamos todas las colecciones para poder trabajar con ellas. Un fallo que cometí, y que después modifique, fue listar cada colección cada vez que me hacía falta, por ejemplo dentro de cada brazo del switch, después me di cuenta que listando al principio del proyecto podía utilizarla varias veces sin problemas.

### *CONFIGURACIÓN DEL MENÚ*

He realizado un menú con un Switch. Contiene 14 opciones que cubren la mayor parte de los requisitos, que podría tener una biblioteca.

```
System.out.println("-----MENÚ-----");
System.out.println("1.Añadir Libro.");
System.out.println("2.Borrar Libro.");
System.out.println("3.Añadir Usuario.");
System.out.println("4.Borrar Usuario.");
System.out.println("5.Realizar un Prestamo.");
System.out.println("6.Realizar una Devolución.");
System.out.println("7.Listar todos los elementos");
System.out.println("8.Ordenar por orden alfabético los libros.");
System.out.println("9.Ordenar por libros disponibles.");
System.out.println("10.Ordenar por los libros prestados.");
System.out.println("11.Listar Usuarios.");
System.out.println("12.Listar los libros leídos de un usuario concreto.");
System.out.println("13. Prestamos de un Usuario");
System.out.println("14.Abandonando Bibliotecario.");
```

Comentaremos cada opción detenidamente.

## OPCIÓN 1: AÑADIR LIBRO

Se añade un nuevo documento a la Colección libro.

```
System.out.println("Has seleccionado Añadir Libro.");

//Tenemos que obtener el título y el autor por variables
System.out.println("Introduce el título del libro");
String nuevoTitulo= teclado.nextLine();
System.out.println("Introduce el autor del libro");
String autor= teclado.nextLine();

//insertamos el nuevoLibro
InsertOneResult nuevoLibro = Catalogo.insertOne(new Document()
    .append("titulo", nuevoTitulo)
    .append("autor", autor)
    .append("prestado", false)//con comillas guardamos un String, sin comillas es boolean
);

break;
```

- 1.- Se insertan en variables los campo de Libro
- 2.- utilizando el método InsertOneResult indicándole colección Catalogo, realizamos un insertOne. Se indica:

- Título
- Autor
- Prestado => false(en deposito)/true(a préstamo)

## OPCIÓN 2: BORRAR LIBRO

Se borra un documento de la Colección libro.

```
System.out.println("Has seleccionado Borrar Libro.");

//título del libro a eliminar
System.out.println("Introduce el título del libro");
String tituloEliminado= teclado.nextLine();

//eliminamos el libro
Catalogo.deleteOne(Filters.eq("titulo", tituloEliminado));

break;
```

- 1.- Se indica el título del libro que queremos eliminar
- 2.- Utilizando el método deleteOne junto con un filtro, que busca dentro de la colección Catalogo alguna coincidencia en el campo "título", si la encuentra elimina el documento.

### OPCIÓN 3: AÑADIR USUARIO

Se añade un nuevo documento a la colección usuario.

```
case "3":
    System.out.println("Has seleccionado Añadir Usuario.");

    //Tenemos que obtener el nombre y el email por variables
    System.out.println("Introduce nombre del nuevo Usuario");
    String nuevoUsuario= teclado.nextLine();
    System.out.println("Introduce el mail");
    String email= teclado.nextLine();

    //insertamos el nuevoLibro
    InsertOneResult nuevoUsuario = Usuarios.insertOne(new Document()
        .append("nombre", nuevoUsuario)
        .append("email", email)
        .append("LibrosLeidos", Arrays.asList())//añade un arrays vacio
    );

    break;
```

1.- Se insertan en variables los campos de Usuario  
2.- utilizando el método InsertOneResult indicándole colección Usuarios, realizamos un insertOne. Se indica:

- nombre
- mail
- LibrosLeidos => utilizamos el método Arrays.asList() añadimos un arrays vacio a nuestro documento.

### 4. ELIMINAR USUARIO

Se borra un documento de la colección usuario.

```
case "4":
    System.out.println("Has seleccionado Borrar Usuario.");

    //nombre del usuario a eliminar a eliminar
    System.out.println("Introduce el nombre del Usuario a eliminar");
    String usuarioEliminado= teclado.nextLine();

    //eliminamos el usuario
    Usuarios.deleteOne(Filters.eq("nombre", usuarioEliminado));//el filt

    break;
```

1.- Se indica el nombre del usuario que queremos eliminar.  
2.- Utilizando el método deleteOne junto con un filtro, que busca dentro de la colección Usuarios alguna coincidencia en el campo “nombre”, si la encuentra elimina el documento.

## 5.REALIZAR PRESTAMOS

Se añade un documento a la Colección prestamos, y se modifica el valor del campo Estado de la Colección libro.

```
case "5":
    System.out.println("Has seleccionado Realizar un Prestamo.");

    //Insertamos el usuario a buscar
    System.out.println("Introduce el nombre del usuario");
    String nombreUsuario=teclado.nextLine();

    //creamos variables
    boolean usuarioCorrecto=false;//indica si el usuario existe
    ObjectId IdUsuario=null;//registra el id del usuario
    boolean disponibilidad = false;//registra true o false cuando el titulo del libro buscado se encuentre
    ObjectId idLibro=null;//registra el id del libro
    String libroPrestamo = null;//libro introducido por el usuario
    String titulo = null;//registremos el titulo del libro

    //recorremos la lista de usuarios, buscando alguna coincidencia
    for(Document Usuario: listaUsuarios) {
        String usuarioBuscado = Usuario.getString("nombre");//guarda en el string el contenido del campo nombre
        //si coincide el nombre introducido por teclado con alguno de la BD, pasa la variable a true
        if(usuarioBuscado.equals(nombreUsuario)) {
            System.out.println("El usuario existe");
            usuarioCorrecto=true;//indica que el usuario existe
            IdUsuario=Usuario.getObjectId("_id");//insertamos en la variable el valor del objeto id
        }
    }

    //continuamos con el proceso de prestamos
    //true significa usuario registrado en la BD
    if(usuarioCorrecto==true) {
        System.out.println("Introduce el nombre del libro que se va a prestar");
        libroPrestamo = teclado.nextLine();

        //recorremos el catalogo de libros buscando un libro que coincida con el introducido por teclado
        for (Document libro : catalogoLibros) {
            String tituloBuscado = libro.getString("titulo");

            //coincide el titulo del documento q se encuentra en el for con el titulo que hemos introducido
            if(tituloBuscado.equals(libroPrestamo)) {
                //guardamos el valor prestado en una variable si es false, el libro no esta prestado
                disponibilidad = libro.getBoolean("prestado");

                //comprobamos si el libro esta prestado o no
                if(disponibilidad==false) {

                    //modificamos el valor de prestado a true
                    Catalogo.updateOne(Filters.eq("titulo", libroPrestamo),
                        new Document("$set", new Document("prestado", true)));

                    idLibro= libro.getObjectId("_id");//registramos el id del libro
                    titulo= libro.getString("titulo");//registramos el titulo

                    //realizamos las modificaciones en la tabla prestamos

                    //insertar un documento
                    LocalDate fechaActual = LocalDate.now(); // Obtiene la fecha y hora actual
                    InsertOneResult nuevoPrestamo = Prestamos.insertOne(new Document()
                        .append("idLibro", idLibro)
                        .append("Titulo", titulo)
                        .append("Usuario", IdUsuario)
                        .append("Nombre", nombreUsuario)
                        .append("FechaPrestamo", fechaActual)
                        .append("FechaDevolucion", "-----")
                        .append("Estado", "A prestamo")
                    );

                } else {
                    System.out.println("El libro está prestado");
                }
            }
        }
    } else {
        System.out.println("Debes dar de alta al usuario");
    }

    //añadimos el libro al arrays de libros del usuario
    if(disponibilidad==false&&usuarioCorrecto==true) { //indica que el prestamo se ha realizado,
        System.out.println("cacarruta");
        //actualizamos usuarios añadiendole el libro que tiene en prestamo
        Usuarios.updateOne(
            Filters.eq("_id", IdUsuario), //filtro en el documento que id sea igual a id del usuario
            Updates.push("LibrosLeidos", libroPrestamo) //push añade al final del array
        );
    }

    break;
```

- 1.- Se inserta el nombre del Usuario a Buscar.
- 2.- Recorremos usando un for each la listaUsuario, buscando una coincidencia en la BD. Si coincide hay una variable usuarioCorrecto que pasa a true y nos permite. Continuar, sino nos indica que el usuario es incorrecto y salimos.
- 3.- Introducimos el libro que queremos sacar de la biblioteca. Debemos comprobar si está disponible(false => se puede prestar o true => está prestado).
- 4.- recorremos la lista catalogoLibros, buscando una coincidencia en el título. Si existe guardamos el valor del campo prestado en una variable denominada disponibilidad. Si esta variable es false continuamos el programa, sino se interrumpe y nos indica que el libro está prestado.
- 5.- Si el libro está en deposito, debemos modificar su estado prestado a true, para ellos utilizamos el método updateOne, junto al uso de un filtro, para buscar el titulo en la colección Catalogo. Una vez lo encuentra mediante un set, modificamos el valor del campo prestado a true.
- 6.- Guardamos en dos variables el id y el del título del libro.
- 7.- Se actualiza la colección préstamos. Primero generamos una fecha para nuestro préstamo usando LocalDate.now(), obtiene la fecha actual. A continuación, usando el InsertOneResult, y la colección Prestamos junto a insertOne, introducimos todas las variables que hemos ido recopilando en estos pasos:
  - Id del libro
  - Titulo del libro
  - Id del usuario
  - Nombre del usuario
  - Fecha de Préstamo => usando LocalDate.now().
  - Fecha de Devolución => estará vacía.
  - Estado => A Préstamo.
- 8.- Si el proceso fue correcto, actualizamos en la colección Usuario el campo LibrosLeidos añadiéndole el título del libro. Para ello utilizamos updateOne.

## 6.-DEVOLUCIONES

Se modifica el valor del campo prestado en la colección libro, y de los valores de los campos Estados y FechaDevolución de la colección prestamos.

```
case "6":
    System.out.println("Has seleccionado Realizar una Devolución.");

    System.out.println("Introduce el libro a devolver");
    String libroDevolucion= teclado.nextLine();

    //se recorre el arrays buscando coincidencias entre el catalogo y el libro introducido por teclado
    for (Document libro : catalogoLibros) {
        String tituloBuscar = libro.getString("titulo");

        //si existe la coincidencia
        if(tituloBuscar.equals(libroDevolucion)) {

            //tenemos que identificar el id del libro;
            ObjectId idLibroDevuelto = libro.getObjectId("_id");//id del libro devuelto

            //recorremos el arrays buscando los prestamos con el id del libro, puede existir varios prestamos
            //uno estará A préstamo, el resto serán antiguos pedidos
            for(Document d:registroPrestamos) {
                ObjectId idObjeto= d.getObjectId("idLibro");//id del libro

                //dentro de los prestamos si el id del libro introducido coincide con el libro del documento
                // y el estado del préstamo es A préstamo pasamos a modificarlo
                if(idObjeto.equals(idLibroDevuelto)&& d.getString("Estado").equals("A préstamo")) {

                    LocalDate fechaActual = LocalDate.now(); // Obtiene la fecha y hora actual

                    //actualizamos el estado del préstamo
                    //primero se modifica la hora, ya que el filtro utilizado lo modificaremos en el
                    Prestamos.updateOne(
                        Filters.eq("Estado", "A préstamo"),
                        Updates.set("FechaDevolucion", fechaActual)//push para los arrays set para
                    );
                    //modificamos el estado del documento
                    Prestamos.updateOne(
                        Filters.eq("Estado", "A préstamo"),
                        Updates.set("Estado", "Devolución")//push para los arrays set para
                    );

                    //actualizamos el estado del libro
                    Catalogo.updateOne(
                        Filters.eq("_id", idLibroDevuelto),
                        Updates.set("prestado", false)//false indica que el libro está disponible
                    );
                }
            }
        }
    }
}
```

- 1.- Se introduce el nombre del libro que se quiere devolver y se guarda en una variable.
- 2.- Se busca el libro dentro del catálogo, para comprobar si existe. Si existe coincidencia, continuamos el programa sino se muestra mensaje de libro no existente en el catálogo.
- 3.- Se guarda en una variable el id del libro.
- 4.- recorremos los prestamos buscando prestamos que contengan el id del libro. Pueden existir prestamos anteriores ya devueltos, con ese mismo id por ello tendremos que filtrar con el préstamo que tenga su estado en A préstamo.
5. una vez localizado el documento con el id del libro y que su estado es A préstamo, pasamos:



- Guardamos la fecha actual en una variable.
- Usando updateOne:
  - o Colección Prestamos
  - o Se actualiza el campo fechaDevolución con la variable fecha guardada anteriormente.
  - o Actualizamos el campo Estado a Devolución.
  - o Colección Libro => Se actualiza el campo prestado a false.

## 7.- LISTAR TODOS LOS LIBROS

Se muestran todos los documentos de la Colección libro.

```
case "7":
    System.out.println("Listar todos los libros.");

    //mostrar por pantalla
    for (Document libro : catalogoLibros) {
        String mostrarTitulo = libro.getString("titulo");
        String mostrarAutor = libro.getString("autor");

        //asi listamos solo los campos titulos y autor
        //System.out.println("Titulo = " + mostrarTitulo + " Autor = " + mostrarAutor );

        // Formatea la salida para que los titulos y autores estén tabulados en dos columnas
        String output = String.format("Titulo: %-40s Autor: %s", mostrarTitulo, mostrarAutor);

        // Imprime la salida formateada
        System.out.println(output);
    }

    break;
```

- 1.- Se recorre el catalogo, se muestra titulo y autor de todos los documentos.

## 8. ORDENA POR ORDEN ALFABETICO LOS LIBROS

Lista por orden alfabético, del campo titulo, los documentos de la Colección libro,

```
case "8":
    System.out.println("Ordenar por orden alfabético los libros");

    //listar todos los elementos de una lista ordenandolos
    FindIterable<Document> librosOrdenados = Catalogo.find().sort(new Document("titulo",1)); //en

    //mostrar por pantalla
    for (Document libro : librosOrdenados) {
        String mostrarTitulo = libro.getString("titulo");
        String mostrarAutor = libro.getString("autor");

        //asi listamos solo los campos titulos y autor
        //System.out.println("Titulo = " + mostrarTitulo + " Autor = " + mostrarAutor );

        // Formatea la salida para que los titulos y autores estén tabulados en dos columnas
        String output = String.format("Titulo: %-40s Autor: %s", mostrarTitulo, mostrarAutor);

        // Imprime la salida formateada
        System.out.println(output);
    }

    break;
```

- 1.- similar a la anterior, con la única diferencia que debemos crear una lista usando sort. Dentro de sort debemos indicar el campo que debe cogerse como referencia para la ordenación y el modo de ordenarse(ascendente o descendente)

## 9.LISTAR TODOS LOS LIBROS DISPONIBLES

Muestra todos los documentos de la Colección libro cuyo valor es false en el campo prestado.

```
boolean disponible;

//mostrar por pantalla
System.out.println("Libros disponibles");

//recorremos el arrays buscando los libros con el campo prestado en false que nos indica que el l
for (Document libro : catalogoLibros) {
    disponible = libro.getBoolean("prestado");

    if(disponible==false) {
        String mostrarTitulo = libro.getString("titulo");//guardamos titulo
        String mostrarAutor = libro.getString("autor");//guardamos autor

        //asi listamos solo los campos titulos y autor
        //System.out.println("Titulo = "+ mostrarTitulo + " Autor = " + mostrarAutor );

        // Formatea la salida para que los titulos y autores estén tabulados en dos columnas
        String output = String.format("Titulo: %-40s Autor: %s", mostrarTitulo, mostrarAutor);//40
        // Imprime la salida formateada
        System.out.println(output);
    }
}

break;
```

1.- Se recorre la lista catalogoLibros, y se guarda en una variable disponible, el valor del campo prestado, si es false, es decir que el libro se encuentra disponible, se muestra por pantalla el título y el autor del documento libro.

## 10.- LISTAR POR LIBROS PRESTADOS

Muestra todos los documentos de la Colección libro cuyo valor es true en el campo prestado.

```
System.out.println("Listar por los libros prestados");

//mostrar por pantalla
System.out.println("Libros prestados a los usuarios");

//recorremos el arrays buscando los libros que se encuentran prestado, campo prestado==true;
for (Document libro : catalogoLibros) {
    disponible = libro.getBoolean("prestado");

    if(disponible==true) {
        String mostrarTitulo = libro.getString("titulo");//guardamos el titulo
        idLibro = libro.getObjectId("_id");//con el titulo buscamos el usuario que lo tiene
        String mostrarUsuario = identificarUsuarioPrestamo(registroPrestamos, idLibro);//función que
        //asi listamos solo los campos titulos y autor
        //System.out.println("Titulo = "+ mostrarTitulo + " Autor = " + mostrarAutor );

        // Formatea la salida para que los titulos y autores estén tabulados en dos columnas
        String output = String.format("Titulo: %-40s Usuario: %s", mostrarTitulo, mostrarUsuario);//
        // Imprime la salida formateada
        System.out.println(output);
    }
}

break;
```

1.- Similar a la anterior, pero se muestra por pantalla los documentos cuyo valor en el campo prestado es true, indica que el libro se encuentra A préstamo.

2.-A tener en cuenta el uso de la función identificarUsuarioPrestamo, que recibe por parámetro una lista y un ObjectId(se comenta en anexo funciones).

## 11.- LISTAR TODOS LOS USUARIOS

Muestra todos los documentos de la colección usuario.

```
System.out.println("Listar Usuarios");

int contador = 1; //índice para los sysos

//mostramos todos los usuarios de la BD
for(Document user: listaUsuarios) {

    System.out.println(contador + ".- " + user.getString("nombre") + ".");
    contador ++;

}

break;
```

1.- Se recorre listaUsuarios, mostrando todos los documentos existentes. Se ha añadido un contador para que sea un poco más atractivo visualmente.

## 12.- LISTAR LOS LIBROS LEIDOS POR UN USUARIO CONCRETO

Muestra todos los valores del campo librosLeidos de la Colección usuario.

```
System.out.println("Listar los libros leídos de un usuario concreto");

System.out.println("Introduce el nombre del usuario a buscar");
String usuarioBus= teclado.nextLine();

//buscamos en la listaUsuarios, alguna coincidencia con el nombre introducido por teclado
for(Document user: listaUsuarios) {

    //si existe la coincidencia
    if(user.get("nombre").equals(usuarioBus)) {

        //guardamos la lista en una variable Lista para poder después mostrarla
        List<String> LibrosLeidos=user.getList("LibrosLeidos", String.class);

        contador=1;
        //se muestra la lista
        for(String a : LibrosLeidos) {
            System.out.println(contador+".- "+a+".");
            contador ++;
        }
    }

}

break;
```

1.- Se introduce el nombre del Usuario.

2.- Se busca en la listaUsuarios alguna coincidencia, en el caso de existir continuamos el programa.

3.- Al estar los libros guardados en un arrays necesitamos guardarlo en una lista, para después poder realizar un for recorriéndolo.

4.- Usando un for, se muestra por pantalla, se ha añadido un contador.

### 13.- PRESTAMOS DE UN USUARIO

Muestra todos los prestamos en los cuales se ha visto involucrado un usuario.

```
System.out.println("Prestamos de un Usuario");
System.out.println("Introduce el nombre del usuario");
String nameUser= teclado.nextLine();

ObjectId usuarioEncontrado= null;

//buscamos algun usuauario que coincida con el introducido por teclado
for(Document d: listaUsuarios) {

    //si existe la coincidencia, guardamos su id
    if(d.get("nombre").equals(nameUser)) {
        usuarioEncontrado = d.getObjectId("_id");//guardamos id en variable
    }

    //si es distinto de null es que existe el usuario
    if(usuarioEncontrado!=null) {
        System.out.println("Usuario : "+ nameUser+ ".");
        contador =1;//para los sysos

        //pasamos por todos los prestamos buscando coincidencias
        for(Document d: registroPrestamos) {

            //usando el compareTo, si es igual a =, son iguales los Id, mostrmos el contenido
            if(usuarioEncontrado.compareTo(d.getObjectId("Usuario"))==0) {
                System.out.println("-----");
                System.out.println(contador+"º Libro.");
                System.out.println("-----");
                ObjectId idBook = d.getObjectId("idLibro");//obtenemos el id del Libro
                System.out.println(tituloPorId(catalogoLibros,idBook));////usamos una función que nos devuelve el titulo
                System.out.println(d.get("Estado"));// indicamos si esta A préstamo o es Devolcuión
                contador++;
            }
        }
    }
    else {
        System.out.println("Usuario no encontrado");
    }
}
```

- 1.-Se introduce por teclado el nombre del usuario.
- 2.- Se busca en la lista si existe una coincidencia, en ese caso se guarda el ObjectId en una variable continuando así el programa, sino se indica que no hay usuario con ese nombre.
- 3.- Se recorre la lista registroPrestamos buscando documentos que contengan el id del usuario, para ello utilizamos un compareTo.
- 4.- Se muestra por pantalla, el título del libro y su estado (A préstamo/Devolcuión). Para obtener el título del libro se utiliza la función tituloPorId que recibe una lista y un ObjectId por parámetro, devolviendo el título del libro. (Se comenta en anexo de funciones).

### 14.- ABANDONANDO BIBLIOTECARIO

Opción seleccionada para abandonar el menú.

## ANEXO FUNCIONES

### tituloPorId

```
/**
 * Busca y devuelve el titulo de un libro dado su ObjectId.
 *
 * @param catalogoLibros Una colección iterable de documentos que representan el catálogo de libros.
 * @param idLibro El ObjectId del libro cuyo titulo se desea buscar.
 * @return El titulo del libro si se encuentra, de lo contrario, null.
 */
private static String tituloPorId(FindIterable<Document> catalogoLibros, ObjectId idLibro) {
    String retorno=null;

    for(Document d: catalogoLibros) {
        if(idLibro.compareTo(d.getObjectId("_id"))==0) {

            retorno=d.getString("titulo");

        }
    }

    return retorno;
}
```

Busca el titulo de un libro buscándolo por su ObjectId. Esta función recibe por parámetro una lista y un ObjectId; devolviendo un String con el nombre del libro si encuentra coincidencia, sino devuelve null.

### idPorTitulo

```
/**
 * Busca y devuelve el ObjectId de un libro dado su Titulo.
 *
 * @param catalogoLibros Una colección iterable de documentos que representan el catálogo de libros.
 * @param titulo String es el titulo del libro cuyo Id queremos buscar.
 * @return el ObjectId se encuentra, de lo contrario, null.
 */
public static ObjectId idPorTitulo(FindIterable<Document> catalogoLibros, String titulo ) {
    ObjectId retorno=null;

    for(Document d: catalogoLibros) {

        if(titulo.equals(d.getString("titulo"))){
            retorno=d.getObjectId("_id");
        }
    }

    return retorno;
}
```

Dado el titulo de un libro devuelve su ObjectId, recibe por parámetro una lista y un String que debe ser el titulo del libro. Sino lo encuentra devuelve null,

### nombrePorId

```
/**
 * Busca y devuelve el nombre de un Usuario dado su ObjectId.
 *
 * @param listaUsuarios Una colección iterable de documentos que representan a los usuarios.
 * @param idUsuario es el ObjectId para el cual buscamos el nombre del usuario que le corresponde.
 * @return un String con el nombre del Usuario si se encuentra, de lo contrario, null.
 */
public static String nombrePorId(FindIterable<Document> listaUsuarios, ObjectId idUsuario) {
    String retorno=null;

    for(Document d: listaUsuarios) {
        if(idUsuario.compareTo(d.getObjectId("_id"))==0) {
            retorno=d.getString("nombre");
        }
    }

    return retorno;
}
```

Esta función dado el ObjectId de un usuario devuelve su nombre. Recibe por parámetros una lista y un ObjectId y devuelve un String con el nombre si existe, sino devuelve null.

### identificarUsuarioPrestamo

```

/**
 * Busca y devuelve el ObjectId de un libro dado su Título.
 *
 * @param registroPrestamos Una colección iterable de documentos que representan el registro de los prestamos
 * @param idLibro ObjectId del libro que el usuario tiene en Prestamo y del cual queremos saber su nombre.
 * @return un String con el nombre del usuario que ha realizado un prestmoa con ese idLibro si se encuentra,
 */
public static String identificarUsuarioPrestamo(FindIterable<Document> registroPrestamos, ObjectId idLibro) {
    String retorno=null;

    for(Document d: registroPrestamos) {
        if(idLibro.equals(d.getObjectId("idLibro"))) {
            retorno=d.getString("Nombre");
        }
    }
}

```

Esta función recibe el ObjectId de un libro y devuelve el nombre del libro que le corresponde. Recibe una lista y un ObjectId por parámetro y devuelve un String, sino lo encuentra devuelve null.

## **7.- BIBLIOGRAFÍA**

<https://www.mongodb.com/>

<https://es.scribd.com/document/224241274/Taller-mongo-java>

<file:///C:/Users/Ismael/Downloads/Java%20y%20MongoDB.pdf>

<http://ualmtorres.github.io/howtos/MongoDBJava/>

[https://www.tutorialspoint.com/mongodb/mongodb\\_java.htm](https://www.tutorialspoint.com/mongodb/mongodb_java.htm)

<https://www.codeandcoke.com/>

<https://josemmsimo.wordpress.com/2014/01/02/conectando-una-aplicacion-java-con-mongodb-en-windows/java>

<https://www.youtube.com/watch?v=pDiqTmTVG9o>

<https://chatgpt.com>