Memoria Practica VII BBDD Objecte-Relacionals

Que

Modifica el programa anterior para que puga almacenar l'informació a alguna Base de Dades orientada a Objectes o noSQL. El nuevo programa he de implementar CRUD para todas las entidades

Para que

Este ejercicio ha resultado útil para aprender y practicar el funcionamiento de los conectores con bases de datos noSQL. También ha sido beneficioso para adquirir conocimientos sobre bases de datos de este tipo, ya que fue necesario informarme sobre como es su funcionamiento. En mi experiencia, la dificultad de este programa reside la gestión por software, además de nuevas implementaciones en el programa. En resumen, la actividad fue útil para comprender mejor el funcionamiento de conectores, así como para aprender y saber que existen bases de datos noSQL, entre otros aspectos.

Pseudocodigo

Conexion

```
Clase Conexion:

    Constante DB = "10813358"
    Constante USER = "10813358"
    Constante PASSWORD = "10813358"
    Constante IP = "10.0.219.21"
    Constante PORT = "27017"
    Constante URL = "mongodb://localhost:27017/"

    Declarar una variable estática MongoClient llamada mongoClient

    Función getConexion:
        Si mongoClient es nulo:
            Asignar a mongoClient la instancia de MongoClients.create(URL)
        Devolver la base de datos obtenida desde mongoClient usando DB

    Función testConexion:
        Intentar realizar lo siguiente:
                        Crear una nueva instancia de MongoClient usando
MongoClients.create(URL)
            Obtener la base de datos desde la instancia de MongoClient
             Obtener el primer nombre de colección para asegurarse de que la
conexión es exitosa
         Capturar cualquier excepción y manejarla imprimiendo un mensaje de
error con Colors.errMsg
        Devolver verdadero si no hay excepciones, falso de lo contrario
```

Gestor

```
Clase Gestor:

Función insertarDocumento(nombreColeccion, documento):
    Intentar:
        Obtener la conexión a la base de datos desde la clase de conexión
        Obtener la colección especificada por nombreColeccion
        Insertar el documento en la colección
    Capturar Excepción:
        Imprimir mensaje de error

Función eliminarDocumento(nombreColeccion, filtro):
    Intentar:
        Obtener la conexión a la base de datos desde la clase de conexión
        Obtener la colección especificada por nombreColeccion
        Eliminar el documento que coincide con el filtro en la colección
    Capturar Excepción:
        Imprimir mensaje de error
```

```
Función updateDocumento(nombreColeccion, filtro, documento):
    Intentar:
        Obtener la conexión a la base de datos desde la clase de conexión
        Obtener la colección especificada por nombreColeccion
         Actualizar el documento que coincide con el filtro en la colección
con los datos del nuevo documento
    Capturar Excepción:
        Imprimir mensaje de error

Función realizarConsultaMongoDB(nombreColeccion, filtro):
    Intentar:
        Obtener la conexión a la base de datos desde la clase de conexión
        Obtener la colección especificada por nombreColeccion
        Realizar la consulta en la colección con el filtro dado
        Retornar el resultado de la consulta
    Capturar Excepción:
        Imprimir mensaje de error
        Retornar null

Función      getAtribute(nombreColeccion,      filtro,      nombreAtributo,
tipoResultado):
      Obtener el resultado de realizarConsultaMongoDB con los parámetros
proporcionados
    Si el resultado no es nulo:
        Obtener el primer documento del resultado
        Si el documento no es nulo:
            Obtener el valor del atributo especificado
    Retornar el resultado obtenido

Función getID(nombreAtributo, atributo, collection):
    Crear un filtro con el nombre y atributo dados
     Obtener el resultado de realizarConsultaMongoDB con la colección y el
filtro
    Si el resultado no es nulo:
        Obtener el primer documento del resultado
        Si el documento no es nulo:
            Obtener el ObjectId del documento bajo la clave "_id"
    Retornar el ObjectId obtenido

Función export(collectionName, path):
    Intentar:
        Crear un nuevo archivo en la ruta especificada
        Obtener la conexión a la base de datos desde la clase de conexión
        Obtener la colección especificada por collectionName
        Obtener todos los documentos de la colección
        Escribir cada documento en el archivo
        Imprimir mensaje de éxito
    Capturar Excepción:
        Imprimir mensaje de error

Función importar(collectionName, path):
    Intentar:
        Verificar si el archivo en la ruta especificada existe
        Obtener la conexión a la base de datos desde la clase de conexión
```

```
        Obtener la conexión a la base de datos desde la clase de conexión
        Obtener la colección especificada por collectionName
        Leer cada línea del archivo
            Parsear cada línea a un documento y realizar una acción de
inserción o actualización
        Imprimir mensaje de éxito
    Capturar Excepción:
        Imprimir mensaje de error

Función crearCollections():
    Intentar:
        Obtener la conexión a la base de datos desde la clase de conexión
        Crear las colecciones si no existen
        Imprimir mensajes de éxito o error según corresponda

Función crearCollectionSiNoExiste(database, collectionNombre):
    Verificar si la coleccion no existe
    Si no existe:
        Crear la colecion
        Imprimir mensaje de éxito

Función existeCollection(database, collectionNombre):
    Iterar sobre las coleccion existentes en la base de datos
    Retornar true si la coleccion existe, de lo contrario, false
```

alumnos

```
Clase Alumnos hereda de Gestor:

Constante collection = "alumnos"
Constante path = "res/alumnos.json"
Instancia de ReadClient llamada rc

Función insertarAlumno(nombre, apellidos, fecha, nia):
    Crear documento alumno con los datos proporcionados
    Llamar a insertarDocumento de la clase padre con la colección "alumnos"
y el documento alumno

Función eliminarAlumno(nia):
    Llamar a eliminarDocumento de la clase padre con la colección "alumnos"
y un filtro por NIA igual a nia

Función obtenerID(nia):
    Llamar a getID de la clase padre con el atributo "nia", el valor nia y
la colección "alumnos"

Función comprobarAlumno(nia):
     Retornar true si obtenerID(nia) no es nulo, de lo contrario, retornar
false

Función pedirNIA(exist):
    Repetir hasta que se ingrese un NIA válido:
        Pedir NIA al usuario con opción de salir si exist es true
```

```
        Si NIA es igual a "0" y exist es true, salir del bucle
        Si NIA no es un número positivo, imprimir mensaje de error
                  Si (exist y no comprobarAlumno(NIA)) o (!exist y
comprobarAlumno(NIA)), imprimir mensaje de error
    Retornar NIA

Función alta():
    Intentar:
        Pedir nombre, apellidos, día, mes, año y NIA al usuario
        Construir fecha y mostrar resumen de datos
        Pedir confirmación al usuario para dar de alta al alumno
        Si la confirmación es "s":
            Llamar a insertarAlumno con los datos proporcionados
            Imprimir mensaje de éxito
        Si la confirmación es "n", imprimir mensaje de cancelación
    Capturar Excepción:
        Imprimir mensaje de error

Función baja():
    Intentar:
        Crear instancia de la clase matriculas llamada matr
        Pedir NIA al usuario con opción de salir
        Si NIA no es igual a "0":
            Obtener ID del alumno con obtenerID(NIA)
            Obtener todas las matrículas asociadas al ID del alumno
            Eliminar cada matrícula con deleteMatricula(matrId)
            Imprimir mensaje de éxito
            Eliminar alumno con deleteAlumno(NIA)
            Imprimir mensaje de éxito
    Capturar Excepción:
        Imprimir mensaje de error

Función mostrarAlumnos():
    Intentar:
        Obtener todos los documentos de la colección "alumnos"
        Para cada documento:
            Imprimir información del alumno (NIA, nombre, apellidos, fecha)
    Capturar Excepción:
        Imprimir mensaje de error

Función modificar(atributo, valor):
    Intentar:
        Pedir NIA al usuario con opción de salir
        Si NIA no es nulo:
               Actualizar el documento del alumno con el atributo y valor
proporcionados
            Imprimir mensaje de éxito
    Capturar Excepción:
        Imprimir mensaje de error
```

modulos

```
Clase Modulos hereda de Gestor:

Constante collection = "modulos"
Constante path = "res/modulos.json"
Instancia de ReadClient llamada rc

Función insertarModulo(nombre):
    Crear documento modulo con el nombre proporcionado
    Llamar a insertarDocumento de la clase padre con la colección "modulos"
y el documento modulo

Función eliminarModulo(nombre):
    Llamar a eliminarDocumento de la clase padre con la colección "modulos"
y un filtro por nombre igual a nombre

Función alta():
    Intentar:
        Pedir nombre de módulo al usuario
        Pedir confirmación al usuario para dar de alta al módulo
        Si la confirmación es "s":
            Llamar a insertarModulo con el nombre proporcionado
            Imprimir mensaje de éxito
        Si la confirmación es "n", imprimir mensaje de cancelación
    Capturar Excepción:
        Imprimir mensaje de error

Función baja():
    Intentar:
        Crear instancia de la clase matriculas llamada matr
        Pedir nombre de módulo al usuario con opción de salir
        Si el nombre no es igual a "0":
            Pedir confirmación al usuario para eliminar el módulo
            Si la confirmación es "s":
                Obtener ID del módulo con getID(nombre)
                Obtener todas las matrículas asociadas al ID del módulo
                Eliminar cada matrícula con deleteMatricula(matrId)
                Imprimir mensaje de éxito
                Eliminar módulo con deleteModulo(nombre)
                Imprimir mensaje de éxito
            Si la confirmación es "n", imprimir mensaje de cancelación
    Capturar Excepción:
        Imprimir mensaje de error

Función obtenerID(nombre):
     Llamar a getID de la clase padre con el atributo "nombre", el valor
nombre y la colección "modulos"

Función comprobarModulo(nombre):
      Retornar true si obtenerID(nombre) no es nulo, de lo contrario,
retornar false

Función pedirNombre(exist):
```

```
Función pedirNombre(exist):
    Repetir hasta que se ingrese un nombre de módulo válido:
         Pedir nombre de módulo al usuario con opción de salir si exist es
true
        Si el nombre es igual a "0" y exist es true, salir del bucle
                Si (exist y no comprobarModulo(nombre)) o (!exist y
comprobarModulo(nombre)), imprimir mensaje de error
    Retornar nombre

Función mostrarModulos():
    Intentar:
        Obtener todos los documentos de la colección "modulos"
        Inicializar contador en 0
        Para cada documento:
            Incrementar contador
            Imprimir información del módulo (contador, nombre)
    Capturar Excepción:
        Imprimir mensaje de error

Función modificar():
    Intentar:
        Pedir nombre antiguo de módulo al usuario
        Pedir nuevo nombre de módulo al usuario
        Si nombre antiguo y nuevo no son nulos:
               Llamar a updateDocumento de la clase padre con la colección
"modulos", un filtro por nombre antiguo y un nuevo documento con el nombre
nuevo
            Imprimir mensaje de éxito
    Capturar Excepción:
        Imprimir mensaje de error
```

matriculas

```
Clase Matriculas hereda de Gestor:

Constante collection = "matriculas"
Constante path = "res/matriculas.json"
Instancia de ReadClient llamada rc
Instancia de Alumnos llamada al
Instancia de Modulos llamada mod

Función insertarMatricula(idAlumno, idModulo, notas):
    Crear documento matricula con _id generado, idAlumno, idModulo y notas
       Llamar a insertarDocumento de la clase padre con la colección
"matriculas" y el documento matricula

Función eliminarMatricula(idMatricula):
       Llamar a eliminarDocumento de la clase padre con la colección
"matriculas" y un filtro por _id igual a idMatricula

Función encontrarIDconIDs(idAlumno, idModulo):
    Crear filtro con idAlumno e idModulo
      Obtener _id con getAtribute de la clase padre usando la colección
```

```
    Obtener "_id" con getAtributo de la clase padre usando la colección
"matriculas", el filtro y "_id" como atributo

Función obtenerIds(exist):
    Repetir hasta que se ingresen datos válidos:
        Obtener idAlumno con getID de la instancia de Alumnos llamada al y
pedirNIA(true)
        Si idAlumno no es nulo:
            Obtener idModulo con getID de la instancia de Modulos llamada
mod y pedirNombre(true)
            Si idModulo no es nulo:
                Obtener idMatricula con encontrarIDconIDs(idAlumno,
idModulo)
                Si (exist y idMatricula es nulo) o (!exist y idMatricula no
es nulo):
                    Retornar un arreglo con idMatricula, idAlumno e
idModulo
                Si no, imprimir mensaje de error
            Si no, imprimir mensaje de cancelación de operación de módulo
        Si no, imprimir mensaje de cancelación de operación de matrícula

Función crearMatricula():
    Intentar:
        Repetir hasta que se desee salir:
            Obtener ids con obtenerIds(true)
            Si idMatricula, idAlumno e idModulo no son nulos:
                Si idMatricula es nulo:
                    Insertar matrícula con insertarMatricula(idAlumno,
idModulo, null)
                    Imprimir mensaje de éxito
                    Salir del bucle
                    Si no, imprimir mensaje de que el alumno ya está
matriculado en este módulo
            Si no, salir del bucle
    Capturar Excepción:
        Imprimir mensaje de error

Función eliminarMatricula():
    Intentar:
        Repetir hasta que se desee salir:
            Obtener ids con obtenerIds(true)
            Si idMatricula, idAlumno e idModulo no son nulos:
                Si idMatricula no es nulo:
                    Eliminar matrícula con eliminarMatricula(idMatricula)
                    Imprimir mensaje de éxito
                    Salir del bucle
                    Si no, imprimir mensaje de que el alumno no está
matriculado en este módulo
            Si no, salir del bucle
    Capturar Excepción:
        Imprimir mensaje de error

Función notasDouble(notasString):
    Si notasString no es nulo y no es "null" ni una cadena vacía:
        Dividir notasString por "#" y convertir cada elemento a Double
```

```
        Retornar arreglo de Double
    Si no, retornar arreglo de Double vacío

Función getNotas(matrID):
      Obtener notas como String usando getAtribute de la clase padre con
colección "matriculas", filtro por _id igual a matrID y atributo "notas"
    Retornar resultado de notasDouble(notasString)

Función modificarNotas():
    Intentar:
        Repetir hasta que se desee salir:
            Obtener ids con obtenerIds(true)
            Si idMatricula, idAlumno e idModulo no son nulos:
                Si idMatricula no es nulo:
                    Obtener notas con modNotas(idMatricula)
                    Crear filtro por _id igual a idMatricula
                    Crear documento update con "notas" igual a notas
                           Llamar a updateDocumento de la clase padre con
colección "matriculas", filtro y update
                    Imprimir mensaje de éxito
                    Salir del bucle
                       Si no, imprimir mensaje de que el alumno no está
matriculado en este módulo
            Si no, salir del bucle
    Capturar Excepción:
        Imprimir mensaje de error

Función notasToString(notasArrayList):
    Crear StringBuilder llamado result
    Para cada nota en notasArrayList:
        Agregar nota seguido de "#" a result
    Si result tiene al menos un carácter:
        Eliminar último carácter de result
    Retornar result como cadena

Función modNotas(matrID):
    Obtener notas como arreglo de Double con getNotas(matrID)
    Crear ArrayList llamado notasArrayList con elementos de notas
    Repetir hasta que se desee salir:
        Mostrar menú de notas con opción de salir
        Pedir posición a modificar
        Si posición es 0, salir del bucle
        Pedir nueva nota
         Si posición es igual al tamaño de notasArrayList más uno, agregar
nueva nota a notasArrayList
        Si no, reemplazar nota en posición-1 con nueva nota
    Retornar resultado de notasToString(notasArrayList)

Función mostrarFormato(alumnoID, moduloID, notas):
    Obtener datos de alumno y módulo con getAtribute de la clase padre
    Mostrar información de alumno y módulo
    Mostrar notas con mostrarNotas(notas)

Función mostrarNotasModulo():
```

```
    Intentar:
        Obtener ids con obtenerIds(true)
        Si idMatricula, idAlumno e idModulo no son nulos:
            Obtener notas con getNotas(idMatricula)
            Mostrar formato con mostrarFormato(idAlumno, idModulo, notas)
        Si no, imprimir mensaje de error
    Capturar Excepción:
        Imprimir mensaje de error

Función mostrarAlumno():
    Intentar:
        Pedir NIA al usuario con pedirNIA(true)
        Si NIA no es nulo:
             Obtener idAlumno con getID de la instancia de Alumnos llamada
al
            Obtener matrículas del alumno con realizarConsultaMongoDB de la
clase padre
            Para cada matrícula en matrículas:
                Obtener idModulo con getAtribute de la clase padre
                Obtener notas con getNotas(matrícula._id)
                     Mostrar formato con mostrarFormato(idAlumno, idModulo,
notas)
        Si no, imprimir mensaje de cancelación
    Capturar Excepción:
        Imprimir mensaje de error
```

menus

```
Clase Menus:

Instancia de ReadClient llamada rc
Instancia de Gestor llamada gs
Instancia de Alumnos llamada gsAl
Instancia de Modulos llamada gsMod
Instancia de Matriculas llamada gsMat

Función mainMenu():
    Repetir hasta que se desee salir:
        Mostrar menú principal
        Pedir opción
        Según la opción elegida:
            Caso 0:
                Imprimir mensaje de cierre
                Salir del bucle
            Caso 1:
                Llamar a menuAlumnos()
                Romper
            Caso 2:
                Llamar a menuModulos()
                Romper
            Caso 3:
                Llamar a menuMatriculas()
```

```
                Romper
            Caso 4:
                Llamar a importar()
                Romper
            Caso 5:
                Llamar a exportar()
                Romper
            Predeterminado:
                Imprimir mensaje de valor no válido
        Fin Según
    Fin Repetir

Función menuAlumnos():
    Repetir hasta que se desee salir:
        Mostrar menú de alumnos
        Pedir opción
        Según la opción elegida:
            Caso 0:
                Imprimir mensaje de salida del menú
                Salir del bucle
            Caso 1:
                Llamar a gsAl.alta()
                Romper
            Caso 2:
                Llamar a gsAl.baja()
                Romper
            Caso 3:
                Llamar a gsAl.mostrarAlumnos()
                Romper
            Caso 4:
                Pedir nuevo nombre
                Llamar a gsAl.modificar("nombre", nuevoNombre)
                Romper
            Caso 5:
                Pedir nuevos apellidos
                Llamar a gsAl.modificar("apellidos", nuevosApellidos)
                Romper
            Caso 6:
                Pedir nuevo día, mes y año de nacimiento
                Formatear fecha
                Llamar a gsAl.modificar("fecha", nuevaFecha)
                Romper
            Caso 7:
                Pedir nuevo NIA y antiguo NIA
                Llamar a gsAl.modificar("nia", nuevoNIA)
                Romper
            Predeterminado:
                Imprimir mensaje de valor no válido
        Fin Según
    Fin Repetir

Función menuModulos():
    Repetir hasta que se desee salir:
        Mostrar menú de módulos
        Pedir opción
```

```
        Pedir opción
        Según la opción elegida:
            Caso 0:
                Imprimir mensaje de salida del menú
                Salir del bucle
            Caso 1:
                Llamar a gsMod.alta()
                Romper
            Caso 2:
                Llamar a gsMod.baja()
                Romper
            Caso 3:
                Llamar a gsMod.mostrarModulos()
                Romper
            Caso 4:
                Llamar a gsMod.modificar()
                Romper
            Predeterminado:
                Imprimir mensaje de valor no válido
        Fin Según
    Fin Repetir

Función menuMatriculas():
    Repetir hasta que se desee salir:
        Mostrar menú de matrículas
        Pedir opción
        Según la opción elegida:
            Caso 0:
                Imprimir mensaje de salida del menú
                Salir del bucle
            Caso 1:
                Llamar a gsMat.crearMatricula()
                Romper
            Caso 2:
                Llamar a gsMat.eliminarMatricula()
                Romper
            Caso 3:
                Llamar a gsMat.modificarNotas()
                Romper
            Caso 4:
                Llamar a gsMat.mostrarNotasModulo()
                Romper
            Caso 5:
                Llamar a gsMat.mostrarAlumno()
                Romper
            Caso 6:
                Llamar a gsMat.mostrarCentro()
                Romper
            Predeterminado:
                Imprimir mensaje de valor no válido
        Fin Según
    Fin Repetir

Función exportar():
    Intentar:
```

```
    Intentar:
        Llamar a gs.export(alumnos.collection, alumnos.path)
        Llamar a gs.export(modulos.collection, modulos.path)
        Llamar a gs.export(matriculas.collection, matriculas.path)
    Capturar Excepción:
        Imprimir mensaje de error

Función importar():
    Intentar:
        Llamar a gs.importar(alumnos.collection, alumnos.path)
        Llamar a gs.importar(modulos.collection, modulos.path)
        Llamar a gs.importar(matriculas.collection, matriculas.path)
    Capturar Excepción:
        Imprimir mensaje de error
```

App

```
Clase App:

    Función main():
        Si Conexion.testConexion() es verdadero:
            Llamar a Gestor.crearCollections()
            Instanciar la clase menus como objeto menus
            Llamar a menus.mainMenu()
        Sino:
             Imprimir mensaje de error indicando que no se pudo iniciar la
conexión
        Fin Si
```

Como

Conexion

Esta clase está diseñada para gestionar la conexión a una base de datos MongoDB. MongoDB es un sistema de gestión de base de datos NoSQL, orientado a documentos. Utiliza un formato de almacenamiento flexible tipo BSON (Binary JSON).

Utilizando las variables constantes proporcionadas, podemos obtener una conexión a la base de datos deseada. En este caso, la clase utiliza el controlador oficial de MongoDB para Java, el cual se encuentra en el paquete `com.mongodb.client`. La conexión a la base de datos se establece mediante la creación de un objeto de tipo `MongoClient`, que es proporcionado por la biblioteca de MongoDB.

La URL de conexión se compone de la dirección IP y el puerto del servidor de MongoDB, así como las credenciales de usuario (nombre de usuario y contraseña) para autenticar la conexión. Para facilitar la creación de la URL de conexión, se proporciona un método `getConexion()` que devuelve un objeto `MongoDatabase`, que representa la conexión establecida con la base de datos.

Además, la clase incluye un método `testConexion()` que intenta establecer una conexión de prueba con la base de datos. En caso de éxito, se realiza una operación sencilla para confirmar la conectividad. En caso de fallo, se maneja la excepción y se imprime un mensaje de error utilizando la clase `Colors` de utilidades.

Gestor

La clase `Gestor` está diseñada para simplificar la gestión de operaciones en una base de datos MongoDB. Utiliza la clase `Conexion` para establecer conexiones MongoDB y proporciona funcionalidades como la inserción, eliminación y actualización de documentos en colecciones específicas. Además, ofrece métodos para realizar consultas, obtener atributos específicos, y exportar e importar datos desde y hacia archivos.

El método `insertarDocumento()` facilita la inserción de documentos en una colección específica de la base de datos MongoDB.

Los **documentos** son la unidad básica de almacenamiento en MongoDB y están compuestos por pares clave-valor. Pueden contener campos anidados y matrices, lo que permite modelar datos de manera flexible.

Una **colección** en MongoDB es un conjunto de documentos almacenados en la base de datos. Se asemeja a una tabla en bases de datos relacionales, pero difiere en el sentido de que no impone un esquema fijo para los documentos.

Se utiliza la clase `MongoCollection` para acceder a la colección deseada y se ejecuta `insertOne()` para agregar el documento.

El método `eliminarDocumento()` permite eliminar documentos de una colección mediante la especificación de un filtro en formato `Document`. Utiliza la función `deleteOne()` de la clase

`MongoCollection` para realizar la operación.

El método `updateDocumento()` simplifica la actualización de documentos en una colección. Se utiliza `updateOne()` con un filtro y un documento de actualización, aplicando la operación `$set` para modificar los valores.

El método `realizarConsultaMongoDB()` se encarga de ejecutar consultas en una colección MongoDB, para ello se obtiene toda laa colleacion y se filtra con la funcion `find()` de `MongoCollection`, para filtrar se necesita un nuevo documento que se pide como parametro que actua como filtro y devuelve un `FindIterable<Document>` que representa el resultado de la consulta. Este objeto es una interfaz proporcionada por el controlador oficial de MongoDB para Java, que forma parte de la API de MongoDB Java. Su función principal es permitir la iteración sobre los documentos resultantes de una consulta.

El método `getAtribute()` facilita la obtención de un atributo específico de un documento en una colección, utilizando un filtro y el nombre del atributo como parametros, este hace uso de las declaraciones de uso generico ya que no se sabe que tipo de objeto va a devolver.

El método `getID()` obtiene el identificador único (`ObjectId`) asociado a un documento en base a un atributo específico y su valor llamando a la función `realizarConsultaMongoDB()`

La funcion `export()` permite exportar datos hacia archivos respectivamente, para facilitar la migración y copia de datos entre instancias de la base de datos. Para ello se compruba si el archivo existe, si no es así se crea un nuevo archivo, luego se conecta obtiene la collecion queriada por su nombre, pedida en los parametros de entrada luego se crea un objeto `FindIterable` para recorrer esta collecion y escribir cada documento en el fichero con la funcion `toJson()` de la clase `Document`.

La funcion `importar()` permite importar datos desde archivos respectivamente, para ello recorre cada linea de el fichero y por cada linea ejecuta la funcion `parse()` esta funcion proviene de la clase `Document` y convierte cada linea (en json) a un nuevo Documento para mas tarde insertar-lo con `insertOne()` o actualizar-lo dependiendo de si existe o no.

La función `crearCollections()` se encarga de crear colecciones específicas en la base de datos MongoDB, como "alumnos", "modulos" y "matriculas", utilizando la función `crearCollectionSiNoExiste()`.

La función `existeCollection()` se encarga de devolver un booleano dependiendo de si la collecion existe o no, para eso pide un objeto `MongoDatabase` y el nombre de la colección, y llama a la función `listCollectionNames()` para obtener una lista de nombres de las colecciones, si el nombre de la funcion coincide con alguno de la lista devuelve verdadero, sino falso.

alumnos

La clase `Alumnos` extiende la funcionalidad de la clase `Gestor` y se enfoca en la gestión de operaciones relacionadas con los alumnos en una base de datos.

El método `insertAlumno()` permite la inserción de un nuevo alumno en la base de datos. Toma como parámetros los datos del alumno (nombre, apellidos, fecha de nacimiento y NIA), crea un documento MongoDB con estos datos y utiliza la función `insertarDocumento()` de la clase padre para realizar la inserción en la colección de alumnos.

El método `deleteAlumno()` elimina un alumno de la base de datos según su NIA. Utiliza la función `eliminarDocumento()` de la clase padre para ejecutar la operación de eliminación en la colección de alumnos.

El método `getID()` devuelve el ID (ObjectId) asociado a un alumno en la base de datos, utilizando la función `getID()` de la clase padre.

El método `comprobarAlumno()` verifica si un alumno con un NIA específico existe en la base de datos. Utiliza la función `getID()` y devuelve `true` si el ID no es nulo, indicando que el alumno existe.

El método `pedirNIA()` solicita al usuario un NIA y realiza la validación necesaria para garantizar que sea un número positivo y único en la base de datos, según el valor del parámetro `exist`. Este método utiliza la función `comprobarAlumno()`.

El método `alta()` solicita al usuario los datos de un nuevo alumno, valida la entrada y llama a `insertAlumno()` para agregar al nuevo alumno a la base de datos.

El método `baja()` solicita al usuario el NIA del alumno a dar de baja, verifica su existencia con `pedirNIA()` y elimina al alumno y sus matrículas asociadas utilizando `deleteAlumno()`, `deleteMatricula()`.

El método `mostrarAlumnos()` imprime por pantalla la información de todos los alumnos presentes en la base de datos, utilizando `realizarConsultaMongoDB()` para obtener los documentos de la colección como un `FindIterable`, luego recorre el objeto y guarda en variables cada campo con `getString()` de la clase Documento para imporimirlo mas tarde con un formato determinado.

El método `modificar()` permite la modificación de un campo específico de un alumno. Solicita al usuario el NIA, verifica su existencia y utiliza `updateDocumento()` de la clase padre para realizar la modificación en la base de datos.

modulos

La clase `Modulos` extiende la funcionalidad de la clase `Gestor` y se enfoca en la gestión de operaciones relacionadas con los módulos en una base de datos MongoDB.

El método `insertModulo()` permite la inserción de un nuevo módulo en la base de datos. Toma como parámetro el nombre del módulo, crea un documento MongoDB con este dato y utiliza la función `insertarDocumento()` de la clase padre (`Gestor`) para realizar la inserción en la colección de módulos.

El método `deleteModulo()` elimina un módulo de la base de datos según su nombre. Utiliza la función `eliminarDocumento()` de la clase padre para ejecutar la operación de eliminación en la colección de módulos.

El método `alta()` solicita al usuario el nombre de un nuevo módulo, valida la entrada y llama a `insertModulo()` para agregar el nuevo módulo a la base de datos.

El método `baja()` solicita al usuario el nombre del módulo a dar de baja, verifica su existencia con `pedirNombre()` y elimina el módulo y sus matrículas asociadas utilizando `deleteModulo(), deleteModulo()`.

La función `getID()` devuelve el ObjectId asociado a un módulo en la base de datos MongoDB, utilizando el nombre del módulo como parámetro y llamando a `super.getID()`.

La función `comprobarModulo()` verifica si un módulo con un nombre específico existe en la base de datos, utilizando la función `getID()` y devolviendo `true` si el ObjectId no es nulo, indicando que el módulo existe.

La función `pedirNombre()` solicita al usuario el nombre de un módulo, con opciones adicionales según el valor del parámetro `exist`. Realiza validaciones y muestra mensajes de error si el módulo ya existe (`exist = true`) o no existe (`exist = false`). Retorna el nombre del módulo validado.

El método `mostrarModulos()` su funcionamiento es similar a el de la clase alumnos.

El método `modificar()` permite la modificación del nombre de un módulo. Solicita al usuario el nombre antiguo y el nuevo, verifica su existencia y utiliza `updateDocumento()` de la clase padre para realizar la modificación en la base de datos.

matriculas

La clase `Matriculas` extiende la funcionalidad de la clase `Gestor` y se especializa en la gestión de operaciones relacionadas con las matrículas de alumnos en módulos, almacenadas en una base de datos MongoDB.

El método `insertMatricula()` permite la inserción de una nueva matrícula en la base de datos. Toma como parámetros los IDs del alumno y del módulo, así como las notas asociadas a la matrícula. Crea un documento MongoDB con esta información y utiliza la función `insertarDocumento()` de la clase padre (`Gestor`) para realizar la inserción en la colección de matrículas.

El método `deleteMatricula()` elimina una matrícula de la base de datos según su ID. Utiliza la función `eliminarDocumento()` de la clase padre para ejecutar la operación de eliminación en la colección de matrículas.

El método `encontrarIDconIDs()` busca el ID de una matrícula a partir de los IDs del alumno y del módulo asociados. Realiza una consulta a la base de datos utilizando estos parámetros y retorna el ID de la matrícula si existe.

El método `encontrarIDconIDs()` busca y retorna el ID de una matrícula en la base de datos, dados los IDs del alumno y del módulo asociados.

El método `obtenerIds()` permite obtener los IDs del alumno y del módulo, solicitando al usuario la introducción del NIA del alumno y del nombre del módulo. Verifica la existencia de la matrícula

con `encontrarIDconIDs()` y retorna un array de ObjectId.

El método `crearMatricula()` facilita la creación de nuevas matrículas. Utiliza `obtenerIds()` para obtener los IDs del alumno y del módulo, y posteriormente invoca `insertMatricula()` para añadir la matrícula a la base de datos.

El método `eliminarMatricula()` guía al usuario para eliminar una matrícula existente. Usa `obtenerIds()` para obtener los IDs del alumno y del módulo, y luego ejecuta `deleteMatricula()` para eliminar la matrícula de la base de datos.

El método `notasDouble()` convierte un String que representa notas separadas por "#" en un array de tipo Double, realizando el análisis de formato necesario para su procesamiento.

El método `getNotas()` obtiene las notas de una matrícula en la base de datos, dado su ID, y retorna un array de tipo `Double` representando las notas.

El método `notasToString()` convierte un array de notas (`Double`) en un formato de String separado por "#", facilitando su almacenamiento en la base de datos.

El método `modNotas()` permite la modificación de las notas asociadas a una matrícula. Muestra las notas actuales, solicita al usuario la posición de la nota a modificar o añadir, y realiza la actualización de notas.

El método `modificarNotas()` guía al usuario para modificar las notas de una matrícula existente. Utiliza `obtenerIds()` para obtener los IDs del alumno y del módulo, y llama a `modNotas()` para realizar la modificación.

El método `mostrarFormato()` imprime por pantalla el formato de presentación de un alumno junto con sus notas en un módulo específico.

Los métodos `mostrarNotasModulo()`, `mostrarAlumno()` y `mostrarCentro()` permiten visualizar las notas de un alumno en un módulo, las notas de todos los alumnos matriculados y todas las matrículas en el centro, respectivamente. Cada uno de estos métodos realiza consultas a la base de datos y muestra la información obtenida de manera formativa.

menus

Estas funciones organizan las opciones disponibles para el usuario en distintos menús y gestionan la ejecución de acciones relacionadas con alumnos, módulos y evaluaciones, conectándose con las clases `alumnos`, `modulos` y `matriculas` para realizar las operaciones correspondientes en la base de datos. Además de las funciones ya explicadas, tiene dos métodos adicionales:

- `exportar()`: llama a la función `export()` con parametros para alumnos, módulos y matrículas.
- `importar()`: llama a la función `importar()` con parametros para alumnos, módulos y matrículas.

App

Esta clase comprueba que la conexión sea correcta llamando a la función `testConexion()` de `Gestor`. Si es correcta, llama a la función `crearCollections()` de `Gestor` para validar que las colecciones existen i si no crearlas, luego llama a `mainMenu()` de la clase `Menu`.

Conclusión

En conclusión, considero que esta actividad ha sido fascinante y ha aportado de manera significativa a mejorar mis habilidades de programación con conectores y bases de datos noSQL. La utilización de nuevas funciones para acceder y modificar la base de datos desde mi programa ha ampliado mi comprensión y destrezas en este ámbito. A lo largo de la actividad, descubrí algunas funciones que no conocía, como la referencia a un constructor en Java `Double[]::new`.

La práctica de diferentes consultas ha sido beneficiosa y ha reforzado mi conocimiento en el manejo de bases de datos. Además, me intriga conocer las diferentes aproximaciones que mis compañeros han tomado para abordar esta actividad, ya que estoy consciente de que hay varias formas de implementarla. Consultar a mis amigos y obtener explicaciones sobre sus enfoques podría proporcionarme valiosas perspectivas y aprender nuevas técnicas.

En general, aunque la actividad fue fácil de entender conceptualmente, la implementación resultó ser desafiante debido a su implementacion de nuevas funciones, la posibilidad de errores y la gestión de un nuevo funcionamiento en la base de dato. Sin embargo, estoy satisfecho con los resultados obtenidos y considero que la dificultad fue proporcional al aprendizaje adquirido.