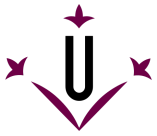


Universitat de Lleida

Grado en Ingeniería informática
Programación II

Práctica 3
"PicoBabel"

Barreto García, Maurys Nicole
Y-7005580R
GPraLab1

INTRODUCCIÓN

Los objetivos de la práctica son los siguientes:

- tratamiento de ficheros secuenciales de texto organizados por líneas.
- tratamiento de ficheros de acceso directo binarios organizados por registros.
- descomposición descendente de acciones y funciones.
- descomposición en clases auxiliares.

El contexto de la práctica es una aplicación para gestionar datos bibliográficos consistentes en libros y sus autores. Para simplificar el problema, los atributos tanto de libros como de autores se han minimizado al máximo; obviamente en un caso real tendríamos muchos más atributos para cada uno de ellos. Las operaciones que consideraremos serán las siguientes:

- dar de alta un nuevo autor.
- dar de alta un nuevo libro.
- pedir información sobre un autor.
- pedir información sobre un libro.

Para guardar de forma permanente la información tanto de autores como de libros, se dispondrá de sendos ficheros binarios de acceso directo. Además, las acciones a realizar vendrán expresadas como líneas en un fichero de texto de movimientos y, al realizar cada operación, tanto de forma exitosa como no, dejaremos constancia en un fichero de bitácora.

ÍNDICE

DESCRIPCIÓN DE LAS CLASES

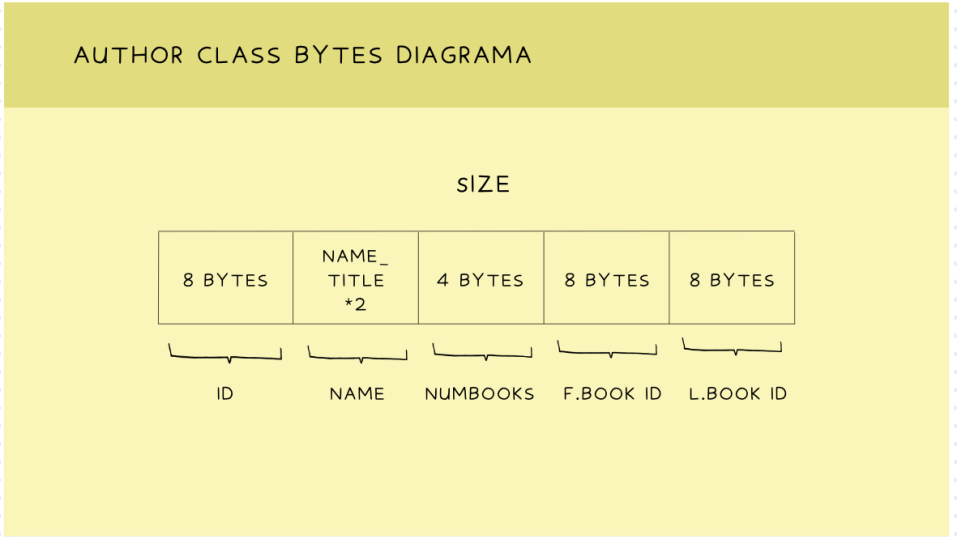
1. La clase Author:

Es la clase que representa la información que se guarda sobre un autor. Podemos clasificar sus atributos en dos grupos:

- Atributos sobre el autor:
  - long id: identificador del autor (comienzan a partir de 1L).
  - String name: nombre del autor.
- Atributos que conectan un autor con sus libros:
  - int numBooks: número de libros que tenemos de ese autor.
  - long firstBookId: identificador del primer libro entrado de ese autor.
  - long lastBookId: identificador del último libro entrado de ese autor.

Además, se tienen las constantes:

- int NAME_LIMIT = 20 que indica el máximo número de caracteres a guardar del nombre cuando se convierte en registro.
- int SIZE = ??? tamaño del registro asociado a un autor.



Con este razonamiento llegamos asignamos a SIZE como 8+NAME_LIMIT*2+ 4+8 *2

Todos los atributos tienen getters y se ha definido un método toString para convertir a cadena de caracteres cualquier instancia de la clase.

Tiene dos constructores:

- public Author(long id, String name, int numBooks, long firstBookId, long lastBookId).
  - inicializa el autor con todos los parámetros indicados.
- public Author(long id, String name)
  - inicializa el libro con los parámetros indicados.

    - inicializa numBooks a 0.
        - inicializa firstBookId a -1L.
        - inicializa lastBookId a -1L.

El primero de ellos se usará cuando disponemos de todos los atributos de un autor; y el segundo cuando, por ejemplo, damos de alta el autor en el sistema y no tiene aún ningún libro asociado.

Además, existe una operación para añadir a un autor un nuevo libro asociado:

- public void addBookId(long idBook).
    - idBook es el identificador del nuevo libro asociado al autor.
    - si es el primer libro del autor (antes el autor no tenía libros), tanto a firstBookId como a lastBookId se les asignará idBook.
    - en caso de que el autor ya tuviera libros, solamente se asignará idBook a lastBookId.
    - en ambos casos se incrementará el número de libros del autor.

Para convertir hacia/desde un array de bytes se dispone de los conocidos métodos:

- public byte[] toBytes() { ??? }
- public static Author fromBytes(byte[] record) { ??? }.

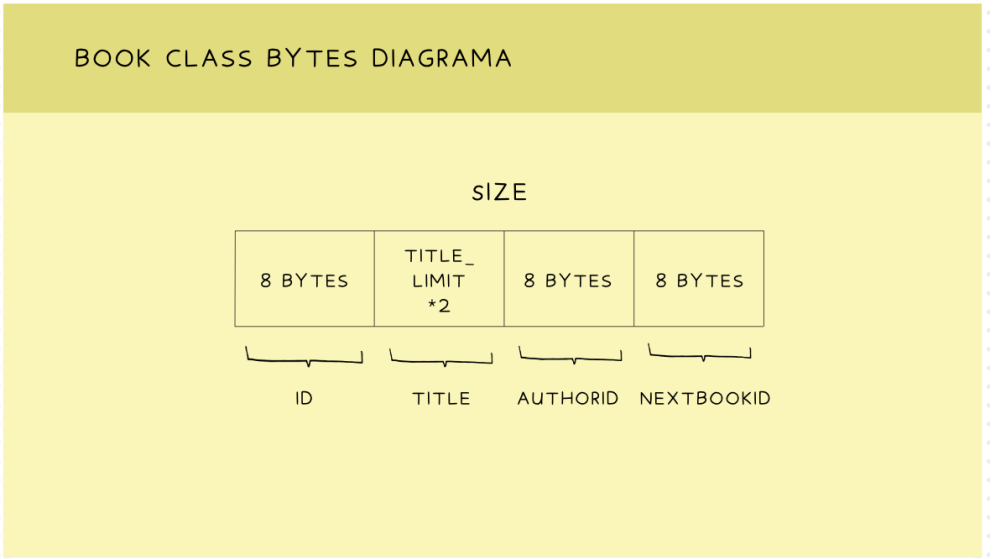
2. La clase Book:

Es la clase que representa la información que se guarda sobre un libro. Podemos clasificar sus atributos en tres grupos:

- Atributos sobre el libro:
    - long id: identificador del libro (comienzan a partir de 1L).
    - String title: título del libro.
- Atributo que conecta un libro con su autor:
    - long authorId: identificador del autor de ese libro.
- Atributo que conecta un libro con el siguiente libro de ese autor.
    - long nextBookId: identificador del siguiente libro del mismo autor; si es el último libro de ese autor su valor será -1L.

Además se tienen las constantes:

- int TITLE_LIMIT = 20 que indica el máximo número de caracteres a guardar del título cuando se convierte en registro.
- int SIZE = ??? tamaño del registro asociado a un libro.



Con este razonamiento llegamos asignamos a SIZE como 8+TITLE_LIMIT*2+8 *2

Todos los atributos tienen getters; nextBookId también tiene un setter; y se ha definido un método toString para convertir a cadena de caracteres cualquier instancia de la clase.

Tiene dos constructores:

- public Book(long id, String title, long authorId, long nextBookId)
    - inicializa el libro con todos los parámetros indicados.

- public Book(long id, String title, long authorId).
    - inicializa el libro con los parámetros indicados.
    - inicializa nextBookId a -1L.

El primero de ellos se usará cuando disponemos de todos los atributos de un libro; y el segundo cuando, por ejemplo, damos de alta el libro en el sistema y éste no tiene aún un siguiente libro.

Para convertir hacia/desde un array de bytes se dispone de los conocidos métodos:

- public byte[] toBytes() { ??? }
- public static Book fromBytes(byte[] record) { ??? }

3. La clase AuthorsFile:

Esta clase servirá para encapsular las operaciones que tienen que acceder al fichero binario de acceso aleatorio que guarda información sobre los autores. Este fichero estará organizado por registros de autores (todos ellos de la misma longitud Author.SIZE) de manera que el primer registro del fichero se corresponde con el autor con identificador 1L, el segundo con el de identificador 2L, etc. Todos los métodos de la clase, constructor incluido, podrán lanzar la excepción IOException.

El constructor de la clase:

- public AuthorsFile(String fname) throws IOException
    - crea la instancia de RandomAccessFile correspondiente al fichero de nombre fname para realizar tanto operaciones de escritura como de lectura.

Operaciones sobre el contenido del fichero y, tal y como hemos hecho en los ejemplos de clase, no se preocuparán de si existen realmente las posiciones de los ficheros a las que intentan acceder.

- public void writeAuthor(Author author) throws IOException
    - escribe los datos del autor en la posición que le corresponde en el fichero según el valor de su identificador (el primer registro del fichero para el autor con identificador 1L, el segundo para el de identificador 2L, etc.)

- public Author readAuthor(long idAuthor) throws IOException
    - devuelve el autor que se encuentra en la posición correspondiente al identificador que se pasa como parámetro
- public long numAuthors() throws IOException
    - devuelve el número de autores que hay en el fichero (recordad que solamente se corresponde con el máximo número de autores si estos se han guardado sin dejar huecos)

Operaciones sobre identificadores:

- public long nextAuthorId() throws IOException
    - devuelve el identificador que le corresponderá al siguiente autor a añadir en el fichero.
    - su valor será uno más que el número de registros existentes actualmente para no dejar huecos en el mismo.
- public boolean isValidId(long idAuthor) throws IOException
    - indica si el identificador dado es válido para el fichero, es decir, su valor está entre 1L y el número de registros del fichero.

Operaciones de gestión del fichero de acceso aleatorio:

- public void reset() throws IOException
    - pone a cero la longitud del fichero de acceso aleatorio correspondiente a los autores (eliminando así su contenido)
- public void close() throws IOException
    - cierra el fichero de acceso aleatorio correspondiente a los autores.

4. La clase BooksFile:

Esta clase servirá para encapsular las operaciones que tienen que acceder al fichero binario de acceso aleatorio que guarda información sobre los libros. Este fichero estará organizado por registros de libros (todos ellos de la misma longitud Book.SIZE) de manera que el primer registro del fichero se corresponde con el libro con identificador 1L, el segundo con el de identificador 2L, etc. Todos los métodos de la clase, constructor incluido, podrán lanzar la excepción IOException.

El constructor de la clase:

- public BooksFile(String fname) throws IOException
    - crea la instancia de RandomAccessFile correspondiente al fichero de nombre fname para realizar tanto operaciones de escritura como de lectura.

Operaciones sobre el contenido del fichero y, tal y como hemos hecho en los ejemplos de clase, no se preocuparán de si existen realmente las posiciones de los ficheros a las que intentan acceder.

- public void writeBook(Book book) throws IOException
    - escribe los datos del libro en la posición que le corresponde en el fichero según el valor de su identificador (el primer registro del fichero para el libro con identificador 1L, el segundo para el de identificador 2L, etc.).

- public Book readBook(long idBook) throws IOException
    - devuelve el libro que se encuentra en la posición correspondiente al identificador que se pasa como parámetro.

- public Book[] getBooksForAuthor(Author author) throws IOException
    - devuelve un array con todos los libros correspondientes al autor.
    - el tamaño del array será el número de libros del autor
    - los libros estarán en el orden en que han estado introducidos

- public long numBooks() throws IOException
    - devuelve el número de libros que hay en el fichero (recordad que solamente se corresponde con el máximo número de libros si estos se han guardado sin dejar huecos).

Operaciones sobre identificadores:

- public long nextBookId() throws IOException
    - devuelve el identificador que le corresponderá al siguiente libro a añadir el fichero.
    - su valor será uno más que el número de registros existentes actualmente para no dejar huecos en el mismo.
- public boolean isValidId(long idBook) throws IOException
    - indica si el identificador dado es válido para el fichero, es decir, su valor está entre 1L y el número de registros del fichero.

Operaciones de gestión del fichero de acceso aleatorio:

- public void reset() throws IOException
    - pone a cero la longitud del fichero de acceso aleatorio correspondiente a los libros (eliminando así su contenido).
- public void close() throws IOException
    - cierra el fichero de acceso aleatorio correspondiente a los libros.

5. La clase PicoBabel:

Las variables de instancia del programa principal son las siguientes:

● movements: fichero de texto de lectura organizado por líneas que contiene las operaciones a realizar.
● logger: instancia de la clase Logger para realizar las operaciones de escritura sobre el fichero de bitácora.
● authorsDB: instancia de la clase AuthorsDB que contiene la información sobre autores.
● booksDB: instancia de la clase BooksDB que contiene la información sobre libros.

MÉTODO PRINCIPAL

Algunas características previas que deberíamos tener en cuenta sobre este método son las siguientes:

● formato del fichero de movimientos.
● tipos de movimientos que pueden realizarse
    ○ casos de error que debemos controlar ○ casos que no se han de controlar y que podemos suponer que son correctos.
● llamadas que se hacen a los métodos de la clase Logger.

- **Fichero de Movimientos**

Contiene las operaciones a realizar sobre los ficheros de datos de autores y de libros. Concretamente, estas operaciones posibles son:

● dar de alta un nuevo autor.
● dar de alta un nuevo libro.
● pedir información sobre un autor.
● pedir información sobre un libro.

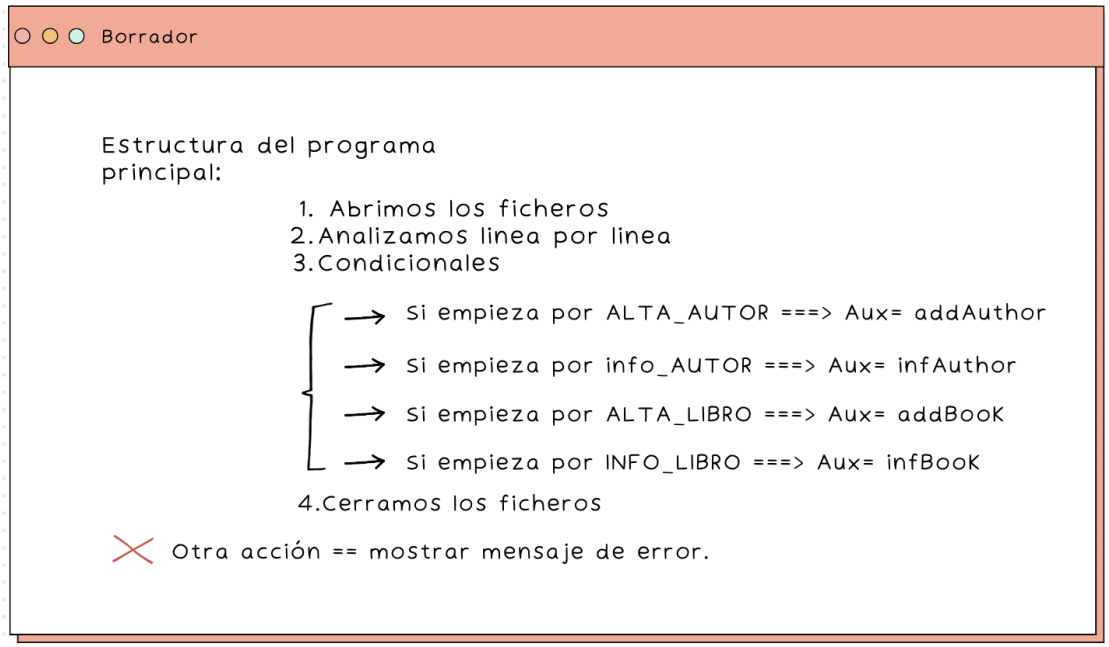
Como en todos los ejemplos que hemos visto tanto en los apuntes como en los problemas, supondremos que el fichero está bien formado y, por tanto, para cada movimiento estarán todos los parámetros y serán de los tipos adecuados (es decir, si han de corresponderse con un número, serán un número).

Eso sí, si detectamos que la operación indicada no es una de las existentes, se anotará tal hecho en el fichero de bitácora utilizando el método *errorUnknownOperation de la clase Logger.*

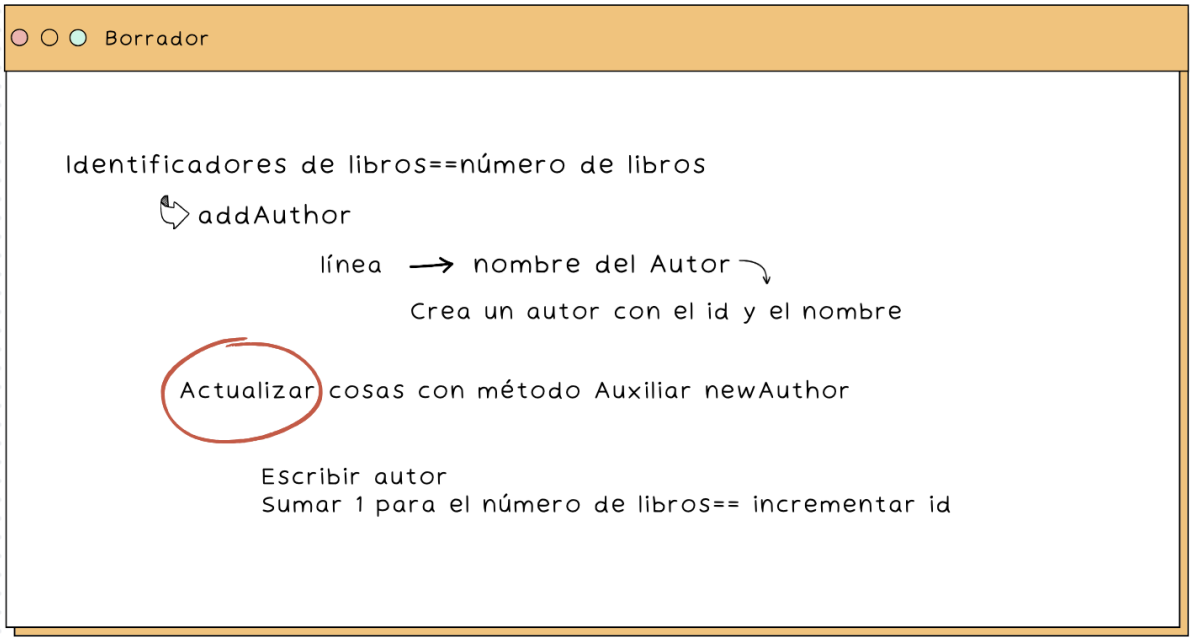
En todas las operaciones, en el momento de detectar un caso de error, se indicará en el fichero de bitácora, y se dejará de ejecutar la operación (por lo que, como máximo, tendremos un error por operación). En caso de que la operación no resulte en un error, también se indicará así en el fichero de bitácora.

- **Resolución.**

Para facilitar la implementación de este método, he separado cada acción para ser manejada de manera individual. La estructura principal del programa es la siguiente:



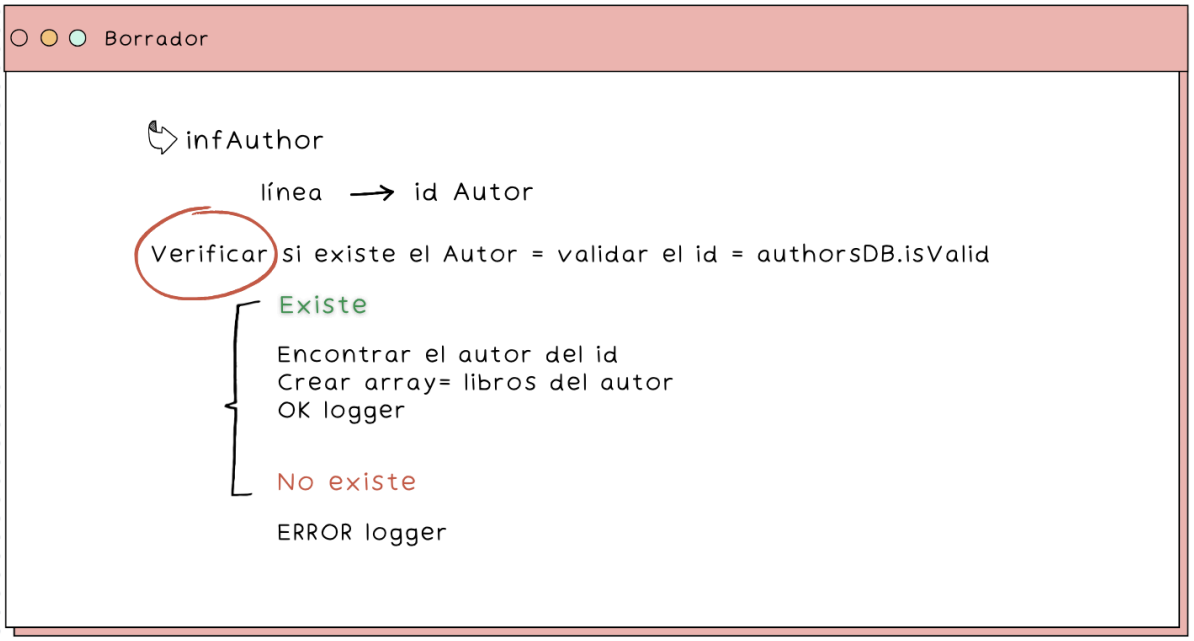
El primer método implementado se ejecutará en el caso de que la línea leída comience con la acción ALTA_AUTOR. Mi borrador para este método es el siguiente:



Es importante tener en cuenta que el identificador de cada libro está relacionado con la cantidad de libros existentes. Por esta razón, inicializaremos una variable para representar este identificador, comenzando en 1, ya que estamos trabajando con el primer libro.

La primera función auxiliar utiliza un StringTokenizer para separar cada característica relevante que nos proporciona la línea que leemos, en este caso, el nombre del autor. Procederemos a crear una nueva instancia de la clase Autor. Este proceso será manejado por el método newAuthor, el cual se encargará de asignar los atributos al autor creado. Luego, registramos esta acción en la bitácora, incrementamos el número de identificador, ya que estamos agregando un nuevo libro, y finalmente, utilizamos la clase logger para indicar que la operación se ha realizado con éxito.

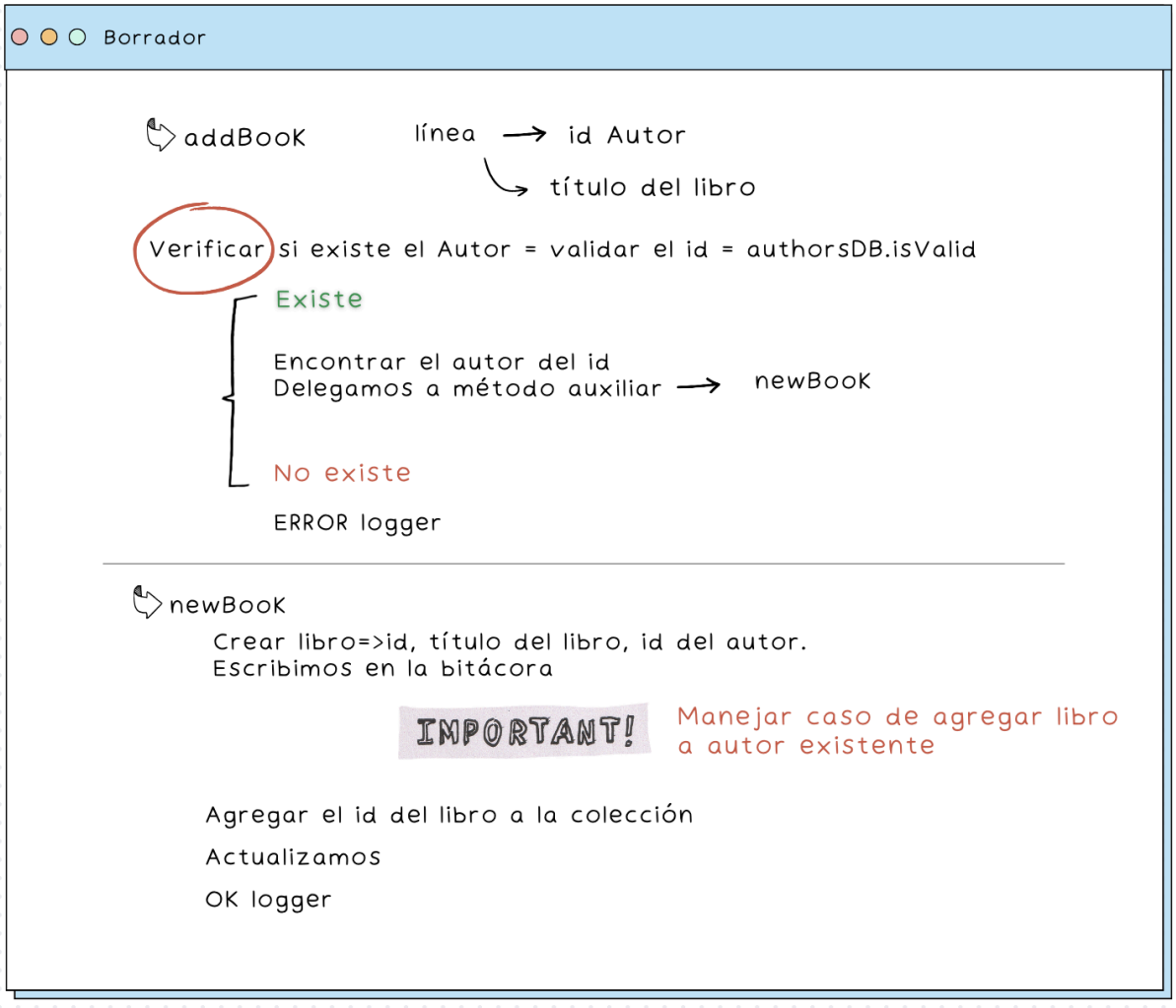
El segundo método implementado se ejecuta si la línea solicita la acción INFO_AUTOR. En este caso, solo tenemos un método auxiliar con la siguiente estructura:



Al igual que en el primer método, comenzamos con un StringTokenizer para asignar valores a los atributos del Author, específicamente su ID. Luego, verificamos si dicho ID existe, es decir, si el autor se encuentra dentro de authors.DB. Si es así, implementaremos unas instrucciones; en caso contrario, mostraremos un mensaje de error.

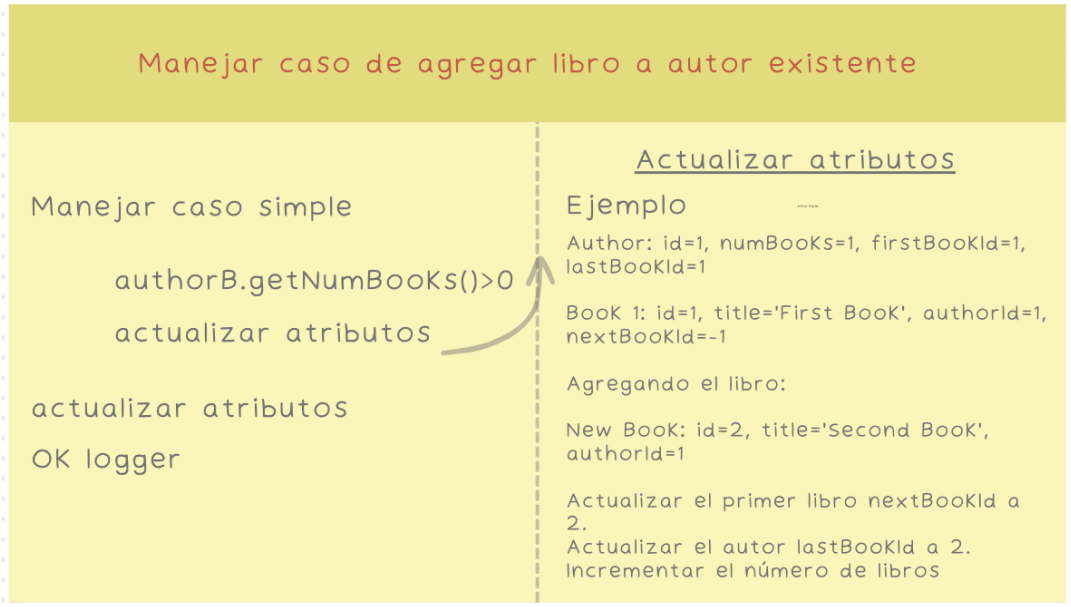
Si el ID es válido, obtendremos al autor correspondiente llamando al método readAuthor(idAuthor) de la clase authors.DB. Finalmente, llamaremos a un método de la clase Books implementado al inicio de la práctica, que nos ayudará a obtener todos los libros de dicho autor. Estos libros se almacenarán en el array authorB. De igual manera, si esta operación se realiza con éxito, indicaremos el ok en la clase logger.

El tercer método auxiliar implementado se centra en manejar los casos en los que la acción es `ALTA_LIBRO`. La estructura del método es la siguiente:



Comenzamos utilizando un `StringTokenizer` para asignar el valor del título del libro y el ID del autor. Al igual que en el apartado anterior, verificamos si dicho ID es válido y procedemos a encontrar al autor correspondiente. Luego, delegamos el siguiente paso al método auxiliar `newBook`.

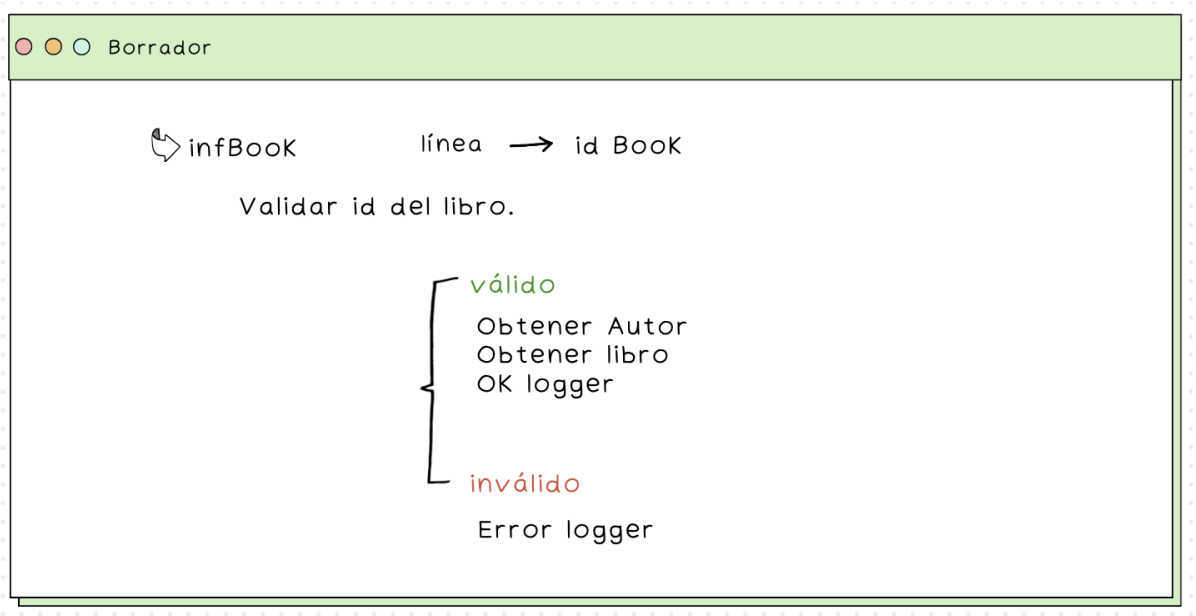
A diferencia de cuando se da de alta a un autor, en el caso de los libros debemos manejar más atributos. Esto significa que debemos considerar un caso cuando se introduce un libro cuyo autor no está en la lista, lo cual sería el escenario más sencillo, y otro caso cuando el autor ya existe y necesitamos agregar el libro a su lista.



Primero, creamos un libro con todos los atributos obtenidos anteriormente y registramos esta acción en la bitácora. Si no es el primer libro del autor, actualizamos el ID del siguiente libro, asegurándonos de mantener el orden en que los libros fueron agregados. Este paso es crucial porque, dependiendo de los identificadores, se pueden modificar atributos como `nextBookId`, que son necesarios para una nueva implementación.

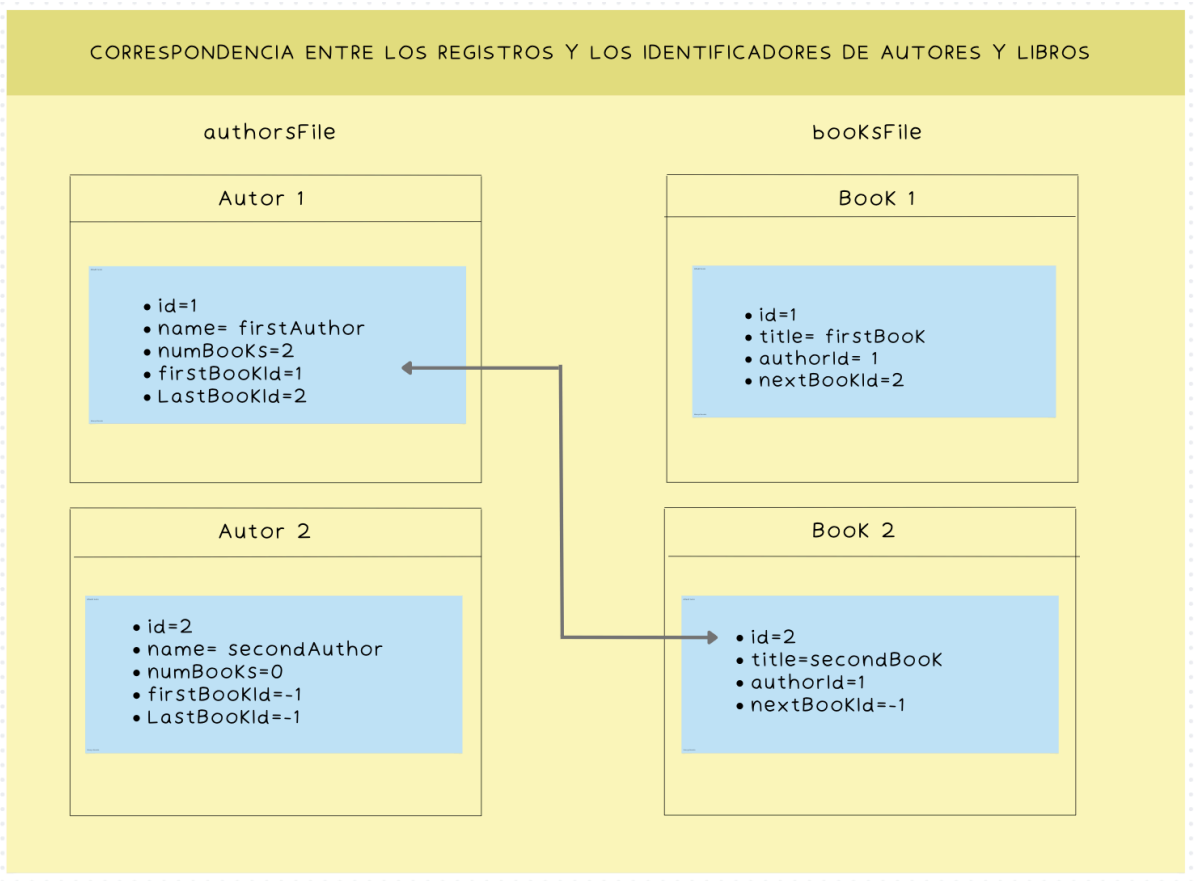
A continuación, agregamos el ID del libro llamando al método correspondiente en la clase `authorB`. Este paso también es relevante, ya que nos ayuda a mantener un recuento de los libros que tiene cada autor. Actualizamos esta información, indicamos el ok en la clase `logger` y finalmente incrementamos el contador de los identificadores de libros.

El último método auxiliar es sencillo.



Utilizamos un `StringTokenizer` para manejar los atributos. Luego, verificamos si el ID del libro es válido, encontramos el libro correspondiente y su autor. Si todo es correcto, indicamos el ok para el correcto funcionamiento del método. En caso contrario, mostramos un mensaje de error.

Después de haber realizado los métodos hemos podido implementar la correspondencia entre los registros y los identificadores de autores y libros, la relación entre las clases y como se manejan la secuencia de libros por el mismo autor. Esto lo podemos visualizar mediante el siguiente diagrama=>



La relación entre las clases Author y Book se puede explicar comenzando con que cada autor tiene la posibilidad de tener múltiples libros, estos son representados mediantes los atributos de la instancia de la clase Author que se encargan de manejar el ID del primer y último libro escrito por el autor.

Además, la clase Book tiene un atributo llamado *nextBookId* que se refiere al próximo libro en la secuencia de libros del mismo autor, permitiendo así una cadena de los mismos.

CONCLUSIÓN

En esta práctica, he aprendido sobre el manejo de ficheros y bytes. Personalmente, ha sido de gran ayuda para entender su correcto funcionamiento. Considero que es muy importante saber cómo guardar en un archivo el resultado de nuestro programa, y esta práctica es un ejercicio esencial para comprender los fundamentos de la manipulación de datos. Estos conocimientos son invaluables para el desarrollo de sistemas más complejos y escalables en el futuro.

Un aspecto interesante que quiero destacar es el reto de gestionar la colección de libros de cada autor. Al principio de la práctica, no tenía muy claro cómo abordar este aspecto. Sin embargo, a medida que implementé los métodos para manejar los libros y los autores, logré entender mejor la relación entre las clases y los respectivos atributos de sus instancias.

BIBLIOGRAFIA

- *RandomAccessFile (Java Platform SE 8)*. (2024, 4 abril). https://docs.oracle.com/javase/8/docs/api/java/io/RandomAccessFile.html
- *BufferedReader (Java SE 20 & JDK 20)*. (2023, 10 julio). https://docs.oracle.com/en/java/javase/20/docs/api/java.base/java/io/BufferedReader.html