PicoBabel - Gestor bibliográfico (v1)

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.

Como es costumbre, en la descripción del enunciado detallaremos los pasos a realizar y el orden en que hay que realizarlos. Además, el proyecto que os proporcionaremos contendrá implementaciones parciales y juegos de prueba de algunas clases.

La clase Author (1 punto)

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)[1]
  - `String name`: nombre del autor

[1] 1L es la forma de expresar la constante 1 como número long (64 bits) ya que 1 sería esa misma constante como número entero (32 bits)

- 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

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) { ??? }`

El primer paso de la solución de la práctica consiste en completar la implementación de esta clase.

De cara a comprobar mínimamente el correcto funcionamiento de la clase se ha creado la clase de pruebas `AuthorTest`.

La clase Book (1 punto)

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

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) { ??? }`

El segundo paso de la solución de la práctica consiste en completar la implementación de esta clase.

De cara a comprobar mínimamente el correcto funcionamiento de la clase se ha creado la clase de pruebas `BookTest`.

La clase AuthorsFile (1 punto)

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 éstos 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

De cara a comprobar posibles errores en la implementación, se dispone de la clase `AuthorsFileTest`.

La clase BooksFile (2 puntos)

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

De cara a comprobar posibles errores en la implementación, se dispone de la clase `BooksFileTest`.

La clase Logger

Esta clase la proporcionamos nosotros y contiene los métodos que escriben, en el fichero de bitácora, tanto los resultados de las operaciones exitosas como los errores que se encuentran en las mismas.

La classe PicoBabel (5 puntos)

Esta es la clase que representará el programa principal y usará todas las clases que se han descrito anteriormente. Su estructura es la siguiente:

```
public class PicoBabel extends CommandLineProgram {

    private static final String MOVEMENTS = "movements.csv";
    private static final String LOG       = "logger.log";
    private static final String AUTHORS   = "authorsDB.dat";
    private static final String BOOKS     = "booksDB.dat";

    private BufferedReader movements;
    private Logger         logger;
    private AuthorsFile    authorsDB;
    private BooksFile      booksDB;

    public static void main(String[] args) {
        new PicoBabel().start(args);
    }

    @Override
    public void run() {
        try {
            openFiles();
            resetData();
            processMovements();
        } catch (IOException e) {
            println("Glups !!!");
            e.printStackTrace();
        } finally {
            try {
                closeFiles();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
```

```
        ....
    }
```

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

El método main

Lo hemos añadido para que el hecho de tener la clase que representa el programa principal fuera del paquete por defecto no presente problemas.

El método run

Más o menos se corresponde con el método run que hemos visto en multitud de ejemplos. Lo único que hemos añadido es el bloque finally, para asegurar que, aunque se produzca un error, se intenten cerrar los ficheros. Esto es especialmente importante para el fichero de bitácora pues, si no se hace así, es posible que lo que se ha escrito en último momento no quede registrado en el fichero (lo que puede complicar saber qué está ocurriendo realmente).
Las llamadas a `printStackTrace()` escriben información sobre qué línea del programa ha llamado a la operación que ha lanzado la excepción.

El método processMovements

La parte más importante de vuestra solución será descomponer en acciones y funciones pequeñas, intentando que cada función realice directamente solamente una tarea y que, por tanto, sea sencilla de programar y de entender.
En las siguientes secciones del enunciado comentaremos los aspectos que debéis de conocer sobre

- formato del fichero de movimientos
- tipos de movimientos que pueden realizarse
    - casos de error que debéis controlar
    - casos que no se han de controlar y que podéis suponer que son correctos
- llamadas que se hacen a los métodos de la clase `Logger`.

El fichero de movimientos

Tal y como se ha comentado anteriormente este fichero 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.

Alta de un autor

El formato de la línea será

```
ALTA_AUTOR,nombre
```

Su ejecución consistirá en:

- obtener el identificador que le corresponde al nuevo autor
- crear una instancia de autor
- guardar la instancia en el fichero de autores
- anotar en la bitácora de que se ha realizado la operación con éxito (método `okNewAuthor` de `Logger`)
- esta operación NO genera nunca casos de error

Alta de un libro

El formato de la línea será

```
ALTA_LIBRO,título,idAutor
```

Su ejecución consistirá en:

- comprobar si `idAutor` es un identificador válido (es decir, se corresponde a algún autor existente)
    - en caso de no serlo, se anota en la bitácora el error producido (método `errorNewBook` de `Logger`) y se finaliza la operación
- obtener el identificador que le corresponde al nuevo libro
- crear una instancia de libro

- guardar la instancia en el fichero de libros
- obtener la instancia de su autor
- añadir el nuevo libro a los libros del autor (pensad en los ajustes que hay que realizar en las información de dicho autor y de alguno de sus libros)
- anotar en la bitácora de que se ha realizado la operación con éxito (método `okNewBook` de `Logger`)

Petición de información de un autor

El formato de la línea será

```
INFO_AUTOR,idAutor
```

Su ejecución consistirá en:

- comprobar si `idAutor` es un identificador válido (es decir, se corresponde a algún autor existente)
    - en caso de no serlo, se anota en la bitácora el error producido (método `errorInfoAuthor` de `Logger`) y se finaliza la operación
- se obtiene el autor del fichero de autores
- se obtienen los libros de ese autor del fichero de libros
- se anota en la bitácora el resultado de la operación (método `okInfoAutor` de `Logger`)

Petición de información de un libro

El formato de la línea será

```
INFO_LIBRO,idLibro
```

Su ejecución consistirá en:

- comprobar si `idLibro` es un identificador válido (es decir, se corresponde a algún libro existente)
    - en caso de no serlo, se anota en la bitácora el error producido (método `errorInfoBook` de `Logger`) y se finaliza la operación
- se obtiene el libro del fichero de libros
- se obtiene su autor del fichero de autores
- se anota en la bitácora el resultado de la operación (método `okInfoBook` de `Logger`)

Un ejemplo de ejecución

Si ejecutamos el programa con el siguiente fichero de movimientos

```
A. ALTA_AUTOR,Autor 1
B. ALTA_AUTOR,Autor 2
C. INFO_AUTOR,1
```

```
D. INFO_AUTOR,3
E. ALTA_LIBRO,Libro 1 autor 1,1
F. ALTA_LIBRO,Libro 2 autor 1,1
G. INFO_LIBRO,1
H. ALTA_LIBRO,Libro 1 autor 4,4
I. INFO_LIBRO,7
J. PATATA,42
K. INFO_AUTOR,1
```

Obtendremos el siguiente fichero de bitácora

```
A. OK ALTA AUTOR Author{id=1, name='Autor 1', numBooks=0,
   firstBookId=-1, lastBookId=-1}
B. OK ALTA AUTOR Author{id=2, name='Autor 2', numBooks=0,
   firstBookId=-1, lastBookId=-1}
C. OK INFO AUTOR Author{id=1, name='Autor 1', numBooks=0,
   firstBookId=-1, lastBookId=-1}
D. ERROR INFO AUTOR (Autor inexistente) 3
E. OK ALTA LIBRO Book{id=1, title='Libro 1 autor 1', authorId=1,
   nextBookId=-1}
F. OK ALTA LIBRO Book{id=2, title='Libro 2 autor 1', authorId=1,
   nextBookId=-1}
G. OK INFO LIBRO Book{id=1, title='Libro 1 autor 1', authorId=1,
   nextBookId=2}
      - Author{id=1, name='Autor 1', numBooks=2, firstBookId=1,
   lastBookId=2}
H. ERROR ALTA LIBRO (Autor inexistente) 4
I. ERROR INFO LIBRO (Libro inexistente) 7
J. ERROR Operación desconocida: PATATA
K. OK INFO AUTOR Author{id=1, name='Autor 1', numBooks=2,
   firstBookId=1, lastBookId=2}
       1 - Book{id=1, title='Libro 1 autor 1', authorId=1,
   nextBookId=2}
       2 - Book{id=2, title='Libro 2 autor 1', authorId=1,
   nextBookId=-1}
```

Hemos añadido una letra indicativa para que sea sencillo cotejar cada línea del fichero movimientos con el texto que se genera en la bitácora.

Formato de la entrega

Un fichero comprimido con **ZIP** que contenga

- directorio del proyecto IntelliJ
    - haced un clean del proyecto antes de comprimirlo para que ocupe el mínimo espacio

- documentación del proyecto (**30% de la nota total**)
    - un informe en **PDF** con
        - portada con el nombre de la práctica, nombre del alumno, dni (o equivalente) y grupo al que pertenece
        - los diagramas correspondientes a los registros de autores y libros indicando los tamaños y posiciones de cada una de las partes del registro
        - los diagramas correspondientes a los ficheros de acceso directo, que muestren la correspondencia entre los registros y los identificadores de autores y libros
        - comentad cómo habéis enfocado la descomposición descendente del método `processMovements`. Una forma de explicarlo es, para cada método de la descomposición, explicar de qué parte del problema se ocupa directamente y qué parte delega en otras operaciones.
            - Por ejemplo, si hacemos la descomposición habitual de tratamiento de las líneas de un fichero de texto, el método `processMovements` lee una a una las líneas del fichero de movimientos y llama a `processMovement` para tratar el movimiento correspondiente a esa línea, ...
    - javadoc para todos los métodos públicos que definàis de las clases `Author`, `Book`, `AuthorsFile` y `BooksFile`.
        - el idioma en que lo escribáis puede ser tanto catalán, castellano o inglés.