Library Manager - El gestor de bibliotecas (v1)

Introducción

Los objetivos de la práctica son

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

El contexto de la práctica es gestionar los datos de una biblioteca, en concreto, permitiendo la realización de las siguientes operaciones:

- dar de alta un libro
- dar de alta un socio
- prestar un libro
- devolver un libro

Para guardar de forma permanente la información tanto de libros como de socios, se dispondrán 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 de este hecho en un fichero de bitácora, que también será un fichero de texto.
Como siempre, en la descripción del enunciado detallaremos los pasos a realizar y el orden en que se han de realizar.

La única "mala noticia" es que, debido al uso de ficheros, dispondremos de muchas menos clases de prueba. No es que no puedan realizarse pruebas con ficheros, pero para poder hacerlo, deberíamos utilizar algunas técnicas que sobrepasan, con mucho, la programación de esta asignatura de primero.

La clase `Book` (1 punto[1])

Es la clase que representa la información que se guarda de un libro. Sus atributos son:

- `long id`: identificador del libro (comienzan a partir de 1L[2])
- `String title`: título del libro
- `long idMember`: identificador del socio que lo tiene o -1L si el libro no está prestado a ningún socio.

El atributo "más complicado" de gestionar es idMember. Con el valor de este atributo sabremos si el libro está libre para ser prestado (porque, cuando no lo tenga nadie, su valor será -1L) y, en caso de estar prestado, sabremos el identificador del socio que lo tiene. Un libro, como máximo, puede estar en cada momento prestado a un único socio.

Además, se tienen las constantes (estáticas) públicas:

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

Todos los atributos tienen getters y, a su vez, el atributo `idMember` también tiene un setter.

Sus constructores son los siguientes:

- `new Book(long id, String title)`
  - inicializa el libros con los parámetros que se le pasan
  - inicializa el atributo `idMember` con el valor -1L
- `new Book(long id, String title, long idMember)`
  - inicializa el libros con los parámetros que se le pasan

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

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

[1] Las puntuaciones de los apartados están expresados en una escala de 10 puntos, aunque la parte de programación es el **80% de la nota total de la práctica**, correspondiendo el otro 20% al informe.
[2] -1L es la constante menos uno pero representada como un long (ya que -1 representa esa misma constante con un int). Cuando se trata de constantes long es una buena práctica añadir el L (aunque, en caso de no hacerlo, ya sabéis que Java haría el cast apropiado) ya que si, por ejemplo, asignara esa constante a una variable entera por error, el compilador detectaría y avisaría del problema.

El primer paso de la solución de la práctica consiste en completar la implementación de esta clase. De cara a comprobar posibles errores disponéis de la clase de test `BookTest`.

La clase `Member` (2 puntos)

Esta clase representará la información que se tiene de un socio de la biblioteca. Sus atributos son:

- `long id`: identificador del socio (comienzan a partir de 1L)
- `String name`: nombre del socio
- `String address`: dirección del socio
- `long[] idBooks`: array de 3 posiciones con los identificadores de los libros (se usará -1L para marcar las posiciones del array que no se corresponden con ningún libro)

En este caso la gestió del atributo que nos servirá para ligar libros y socios se complicará un poco ya que en nuestra biblioteca, un socio puede tener hasta tres libros en préstamo. Eso quiere decir que ya no nos servirá un único atributo, sino que usaremos un array de tres posiciones. Fijaos que, en dicho array, las posiciones que no contengan -1L se corresponderán a los identificadores de los libros que tiene el socio.

Además, se tienen las constantes (estáticas) públicas:

- `NAME_LIMIT = 20` que indica el máximo número de caracteres a guardar del nombre cuando se convierte en registro
- `ADDRESS_LIMIT = 30` que indica el máximo número de caracteres a guardar de la dirección cuando se convierte en registro
- `SIZE = ???` tamaño del registro asociado a un socio

Se dispone de getters para los atributos `id`, `name` y `address` y de las siguientes funciones para manipular el array de identificadores de libros:

- `public int countBooks()`
  - retorna el número de libros que tiene el socio en préstamo
- `public boolean canBorrow()`
  - retorna un booleano indicando si el socio puede pedir en préstamo un libro ya que no ha llegado al máximo que es 3
- `public void addBook(long idBook)`
  - añade el libro con ese identificador a los que se le han prestado al socio
  - en caso de que el socio ya tuviera el máximo de libros, la operación no hace nada
- `public void removeBook(long idBook)`
  - elimina el libro de los que tiene en préstamo el socio
  - si el libro no es uno de ellos, la operación no hace nada
- `public boolean containsBook(long idBook)`
  - retorna un booleano indicando si el libro con código `idBook` es uno de los que tiene el socio en préstamo.

Sus constructores son

- `new Member(long id, String name, String address)`
  - inicializa los atributos con los valores dados
  - crea el array de 3 posiciones `idBooks` e inicializa sus posiciones con el valor -1L
- `new Member(long id, String name, String address, long[] idBooks)`
  - inicializa los atributos con los valores dados

Y, como es de esperar, para convertir hacia/desde un array de bytes se dispone de los conocidos métodos

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

El segundo paso de la solución de la práctica consiste en completar la implementación de esta clase.
De cara a comprobar posibles errores disponéis de la clase de test `MemberTest`.

La clase `BooksFile` (1 punto)

Esta clase servirá para encapsular las operaciones que tienen que ver con el fichero de acceso directo de libros.
Este fichero estará organizado por registros de libros (con longitud Book.SIZE) de manera que el primer registro del fichero se corresponda con el libro con identificador 1, el segundo con el de identificador 2, etc, etc.
Todos los métodos de la clase, constructor incluido, podrán lanzar la excepción `IOException` y no se han de preocupar en ningún momento de si realmente las posiciones del fichero existen (tal y como hemos hecho en todos los ejemplos y problemas).

- `new BooksFile(String fname) throws IOException`
  - crea la instancia de `RandomAccessFile` correspondiente al fichero que tiene como nombre `fname`
- `public void clear() throws IOException`
  - pone a cero la longitud del fichero de libros
- `public Book read(long id) throws IOException`
  - lee el libro del fichero que tiene por `id` el pasado por parámetro
- `public void write(Book book) throws IOException`
  - escribe los datos del libro en las posiciones correspondientes del fichero
- `public long length() throws IOException`
  - indica el número de libros que contiene el fichero (obviamente suponiendo que no hay huecos entre los registros)
- `public void close() throws IOException`
  - cierra el fichero de acceso aleatorio correspondiente a los libros

De cara a comprobar posibles errores disponéis de la clase de test `BooksFileTest`.

La clase `MembersFile` (1 punto)

Esta clase es exactamente como la anterior, solamente que esta vez el fichero de acceso directo contiene socios en vez de libros. Los métodos son equivalentes a los de la clase anterior y sus nombres y parámetros los encontraréis en el proyecto que os proporcionamos.

De cara a comprobar posibles errores disponéis de la clase de test `MembersFileTest`.

La clase `LogFile`

Esta clase os la proporcionamos nosotros y es la que contiene los métodos para escribir los mensajes de OK/ERROR en el fichero de bitácora tras la ejecución de cada una de las líneas del fichero.

La clase `LibraryManager` (5 puntos)

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

```
public class LibraryManager extends CommandLineProgram {

    private static final String BOOKS     = "llibres.dat";
    private static final String MEMBERS   = "socis.dat";
    private static final String LOG       = "manager.log";
    private static final String MOVEMENTS = "movements.csv";

    private BufferedReader movements;

    private BooksFile booksFile;
    private MembersFile membersFile;
    private LogFile logFile;

    private int currentMovementLine;

    public void run() {
        try {
            openFiles();
            processMovements();
            closeFiles();
        } catch (IOException ex) {
            println("ERROR something about the files");
```

```
            }
        }
        ...
    }
```

Las variables de instancia del programa principal son las siguientes:

- `movements`: fichero de texto de lectura organizado por líneas que indica las operaciones a realizar
- `booksFile`: instancia de la clase `BooksFile`, comentada anteriormente, que representa los datos sobre los libros
- `membersFile`: instancia de la clase `MembersFile`, comentada anteriormente, que representa los datos sobre los socios
- `logFile`: instancia de la clase LogFile, comentada anteriormente, que representa el fichero de bitácora
- `movementNumber`: número de línea que estamos tratando del fichero de movimientos (y que se usará al escribir la línea correspondiente del fichero de bitácora)

El método `processMovements`

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

- formato del fichero de movimientos
- tipos de movimientos que pueden realizarse
    - casos de error
- formato del fichero de bitácora

El fichero de movimientos

Tal y como se ha comentado anteriormente este fichero contiene las operaciones a realizar sobre los ficheros de datos de libros y socios. En concreto, las operaciones posibles son:

- alta de un nuevo libro
- alta de un nuevo socio
- préstamo de un libro a un socio
- devolución de 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).
En todas las operaciones, en el momento de detectar un caso de error, se indicará en error en el fichero de bitácora, y se dejará de ejecutar la operación (por lo que, como máximo, tendremos **un error por operación**).

Alta de un libro

El formato de la línea será

`ALTALIBRO,título`

Su ejecución consistirá en

- obtener el valor del `título` (`String`)
- calcular el `id` que le corresponderá (en función del contenido del fichero de libros de manera que el nuevo libro quede al final, sin dejar hueco alguno)
- crear la instancia de libro con esos datos (`id` y `título`)
- guardar el nuevo libro en el fichero de libros
- escribir en el fichero de bitácora que se ha realizado una alta de libro

Alta de un socio

El formato de la línea será

`ALTASOCIO,nombre,dirección`

Su ejecución consistirá en

- obtener los valores de `nombre` y `dirección` (`Strings`)
- calcular el `id` que le corresponderá (en función del contenido del fichero de socios de manera que el nuevo socio quede al final, sin dejar hueco alguno)
- crear la instancia de socio con esos datos (`id`, `nombre` y `dirección`)
- guardar el nuevo socio en el fichero de socios
- escribir en el fichero de bitácora que se ha realizado una alta de socio

Préstamo de un libro a un socio

El formato de la línea será

`PRÉSTAMO,idLibro,idSocio`

Su ejecución consistirá en

- obtener los valores de `idLibro` e `idSocio` (`longs`)
- comprobar que `idLibro` es un identificador válido correspondiente a un libro
    - si no lo es indicar el error en el fichero de bitácora
- comprobar que `idSocio` es un identificador válido correspondiente a un socio
    - si no lo es indicar el error en el fichero de bitácora
- obtener el libro correspondiente a ese código
    - si el libro ya está prestado indicar en el fichero de bitácora el error
- obtener el socio correspondiente a ese código
    - si el socio ya tiene el máximo de libros indicar el error
- realizar el préstamo y actualizar los datos de libro y de socio en los ficheros respectivos
- indicar en el fichero de bitácora que se ha podido realizar el préstamo

Devolución de un libro

El formato de la línea será

`DEVOLUCIÓN,idLibro`

Su ejecución consistirá en

- obtener el valor de `idLibro` (`long`)
- comprobar que `idLibro` es un identificador válido correspondiente a un libro
    - si no lo es indicar el error en el fichero de bitácora
- obtener el libro correspondiente a ese código
    - si el libro no está prestado indicar en el fichero de bitácora el error
- obtener el `idSocio` que tiene el libro
- comprobar que idSocio es un identificador válido correspondiente a un socio
    - si no lo es indicar el error en el fichero de bitácora
- obtener el socio correspondiente a ese código
    - si el socio no tuviera en préstamo el libro indicar el error
- realizar la devolución y actualizar los datos de libro y de socio en los ficheros respectivos
- indicar en el fichero de bitácora que se ha podido realizar la devolución

El fichero de bitácora

Los mensajes en el fichero de bitácora tienen la forma:

```
ERROR(#línea): mensaje
OK(#líniea): mensaje
```

Por ejemplo,

```
OK(1): alta de libro con identificador 45
OK(2): alta de socio con identificador 12
ERROR(3): préstamo: no existe libro con código 50
```

Puntuación de este apartado

- Diseño descendente (descomposición en acciones/funciones) (1,5 puntos)
- Lectura del fichero de movimientos (0,5 puntos)
- Separación de la línea (0,5 puntos)
- Movimientos (2,5 puntos)
    - ALTALIBRO (0,5 punto)
    - ALTASOCIO (0,5 punto)
    - PRÉSTAMO (0,75 puntos)
    - DEVOLUCIÓN (0,75 puntos)

Formato de la entrega

Un fichero comprimido con **ZIP** que contenga

- directorio del proyecto IntelliJ
    - eliminad el directorio out del proyecto antes de comprimirlo para que ocupe el mínimo espacio
- documentación del proyecto (**20% de la nota tota**l)
    - un informe en **PDF** (2 ó 3 páginas contando la portada) con
        - portada con el nombre de la práctica, nombre del alumno, DNI (o equivalente), grupo al que pertenece
        - los diagramas correspondientes a los registros de socios y libros, indicando los tamaños 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 libros y socios
        - comentad cómo habéis enfocado la descomposición descendente del método processMovements. Podéis ayudaros de esquemas, diagramas, etc. pero no uséis código (ése ya está en el listado)
    - **javadoc** para todos lo métodos públicos que definái.s