Library Project: Book.java

In the project, you will develop four classes called Book, Reader, Shelf, and Library. These classes will be used to store, manipulate, and analyze information stored in the library.

This file details the creation of the Book.java class. The UML is listed below.

All the fields have getters and setters. It is also important to note that the class has a hashCode() method and an equals() method.

The constant fields are used to index the values from the library00.csv files as seen below.

Field Details

Constant Fields

ISBN_
TITLE_
SUBJECT_
PAGE_COUNT_
AUTHOR_
DUE_DATE_

These fields will hold an integer that will be used to index the values shown below. The values will be read from a String array that results from using the split() method on the comma separated Strings shown below.
For help with setting up a LocalDate: check the Java API

Library00.csv

```
1337,Headfirst Java,education,1337,Grady Booch,0000
42-w-87,Hitchhikers Guide To the Galaxy,sci-fi,42,Douglas Adams,0000
5297,Count of Monte Cristo,Adventure,999,Alexandre Dumas,0000
42-w-87,Hitchhikers Guide To the Galaxy,sci-fi,42,Douglas Adams,0000
1337,Headfirst Java,education,1337,Grady Booch,0000
34-w-34,Dune,sci-fi,235,Frank Herbert,0000
```

Method Details

equals()

Use inteliJ to auto generate this.
The equals method will compare everything **except** the dueDate.

hashCode()

Use inteliJ to auto generate this.
The hashCode method will compare everything **except** the dueDate.

toString()

The toString will concatenate the title, author, and ISBN in the following order:
`[Title] by [Author] ISBN: [isbn]`

E.g.
`Headfirst Java by Grady Booch ISBN: 1337`

Testing

Each class in this project will require unit tests. I will provide less and less instruction for each class as I want you to develop the ability to write tests on your own. The nice thing is, most tests are going to be pretty similar.

A general rule to follow when writing unit tests is to build up based on 'knowns'. What I mean is:
Does the class instantiate?
Once it does we can test the fields
Once we can ensure the fields are set we can test for accuracy
Then we can test for mutability
Then we can test for functionality

Creation ⇒ Instantiation ⇒ Mutation ⇒ Evaluation[1]

Constructor Test

For help with setting up a LocalDate: check the Java API

1. Create a Book object and set it to null.
2. Assert that it is null
    a. This will ALWAYS pass
3. Instantiate the Book with dummy or empty parameters (i.e. "" for a String)
    a. book = new Book()
4. Assert that it is not null

Field Setting and Getter Test

1. Create variables for each parameter that will be passed in to the constructor.
2. Use AssertEquals for each getter to ensure that the values are set accordingly
    a. This has saved my bacon on more than one occasion when I had the parameters in the wrong order

Equality Test

1. Create an instance of a Book with given values as we did in Field Setting and Getter Test
2. Create an instance of a Book with DIFFERENT values
3. Ensure that they are NOT equal
4. Create a new instance of Book with the same values as one of the previous
5. Assert that they are equal
    a. Note: DO NOT test this:

```
Book book1 = New Book({Parameters});
Book book2 = book1;
```

[1] I know that doesn't quite work but I like the alliterative sound of it.

Setter Test

We now know our getters work. Test the setters.

1. Instantiate a Book like we did in Field setting and Getter Test
2. Declare variables with new values for each parameter
3. Set the values to the new parameter
4. Assert that the values are **not** equal to the old value
5. Assert that the values **ARE** equal to the NEW value

UML Diagram of Book.java