CS 4743
Applied Software Engineering
Spring 2017

Assignment 4: Record Locking and the Book Domain Model
Due: Monday, Mar. 6th 11:59pm

BACKGROUND:
You may have noticed by now that when you run more than one instance of your program simultaneously and two or more instances change the same record, all but one instance lose their changes without any message to the user. This is clearly not a desirable feature! It is very important in enterprise software that when the system makes it appear that a user's changes have saved, that they have indeed saved. Or at least the software must tell the user that it could not save the changes because someone else beat them to it.

The underlying problem her is concurrency management, but not at the thread level. It has to be at the process level. We looked at 2 different techniques for process-level concurrency management: pessimistic record locking and optimistic record locking. In this assignment, we will implement an optimistic locking technique on our Author Detail View (the GUI responsible for editing existing authors) to properly manage the issue of concurrent changes between client instances to the same Author database record.

We will also implement the book database table, domain model, list view/controller, and detail view/controller. This a very straightforward change and we can use much of our Author code as a basis for the Book code (be very careful with copy and paste though!). One very important difference with a Book though is its relationship with an Author and you must be sure to provide an easy way for the user to specify which Author has written the currently viewed Book (a Combo List is a good choice).

TASKS:
1. Implement an **optimistic** record locking mechanism on the Author Detail View.
I suggest modifying your author database table and model to include a last modified time stamp that is fetched and stored in your Author object in the gateway's getAuthor method(s). The modification to the author database table should be a column called "**last_modified**", of type **TIMESTAMP**. You can also set the Attributes value of last_modified to "on update CURRENT_TIMESTAMP", which will tell MySQL to automatically update last_modified to the current server date/time whenever the record is changed. The modification to the Author class should be a **lastModified** member variable of type **LocalDateTime** (requires Java SE 8, which we are using).

Note that JDBC does not directly support LocalDateTime as a get method, but it does support a Timestamp get method, which has methods for conversion. For example, to set the lastModified value on an author object to the value in my resultset record, I could use the following statement:

```
myAuthor.setLastModified(myResultset.getTimestamp("last_modified").toL
ocalDateTime());
```

For completeness, I suggest initializing lastModified to null and when a new Author is inserted into the database via the gateway, the gateway will update the model's lastModified field to the author last_modified column value for the newly inserted record.

Your gateway, before executing the update query, can compare the model's timestamp with the DB's timestamp for that record, and if they are different, throw an exception that the view displays to the user, asking the user to go Back to the Author List to fetch a fresh copy of the Author. If they are the same, the gateway continues with saving the modified model data.

Lastly, if a save is successful, you need to update the model's last modified variable to the new last modified timestamp from the database (updating the record should also update the last modified field in the database).

2. Create your **book** database table in your MySQL database assigned to you for this class. Your fields should be:

- id : int , primary key, auto-incrementing
- title : varchar(100)
- publisher : varchar(100)
- date_published : date
- summary : text (up to 64K characters)
- author_id : int (foreign key to author.id field)
- last_modified : timestamp (set Attributes to "on update CURRENT_TIMESTAMP")

Also, be sure to insert a couple of records into your table for testing.

3. Implement the **Book** model with the following member variables (use appropriate data types):

- id : int
- title : String
- publisher : String
- datePublished : String
- summary : String
- author : Author
- lastModified : LocalDateTime

The only above member variable that may need explaining is the author variable. It is an Author object corresponding to the author that wrote the book (we assume that each book has one and only one author).

You should have at least a no argument constructor that initializes the instance variables appropriately, and getters/setters for all member variables.

4. Implement the BookTableGateway and a getBooks method that returns a List<Book>. All JDBC queries in this assignment MUST USE PARAMETERIZED QUERIES, even if there are no parameters to the query.

Note that your getBooks methods will have to instantiate a Book object, which includes an Author object. You can do this a few different ways:
One approach is to use a JOIN query that pulls both the author information and the book information together in a single record in the resultset. To create a Book object, first instantiate an Author object from the author part of the record. Then instantiate a Book object, set all member variables to the appropriate field data, and set the author member variable to the Author object you already instantiated.
Another approach is to have your book gateway use the author table gateway and have it return an Author instance for the given author_id stored in the book record. That is a simpler approach that requires less code, but you are then making one of your gateways dependent upon another and causing multiple trips to the database.

5. Create a new menu called "Books" and a menu item called "Books List". Then create the BookListView and Controller. The BookListController should be passed a collection of Book models in its constructor. The BookListView should display a ListView (or feel free to use a TableView or other **suitable** JavaFX GUI component for showing a list of items) showing Author First and Last Name and the Book Title. As with the AuthorListView, double-clicking a Book in the ListView should open it in the BookDetailView (mentioned below).

Be sure to also implement a Delete button on the BookListView that deletes the selected book after first prompting the user if he/she really wishes to delete.

6. Add validators and a save method for your Book model. The validator business rules are:

a. an id must be >= 0 (0 should indicate a new Book to be inserted)
b. title cannot be blank and no more than 100 characters
c. publisher cannot be blank and no more than 100 characters
d. summary cannot be blank
e. author must a valid author (e.g., the corresponding author id must be > 0)

Implement these business rules as validation methods that return true or false if the respective field value is valid.

Next, implement all of the save functionality to update a Book that already exists in the database or inserts the Book if it does not exist. Be sure to insert corresponding audit trail entries whenever a book is inserted or changed. The audit_trail record_type for a book is 'B'.
Also be sure to implement your gateway update and insert methods using transactions, similar to your author gateway methods.

7. Create a menu item called “Add Book” in your menu. Then implement your BookDetailView and Controller. The BookDetailViewController should be passed a Book model and a List of Author models (to populate a drop-down list of Authors).

Similar to the Author Detail View, clicking “Add Book” from the menu should show the Book Detail View with a blank Book model. Double-clicking a book in the Book List View should show the Book Detail View with the selected Book model loading into the view.

What makes the Book Detail View really different is that you will have a drop-down list containing all Authors, from which the user may select the appropriate author of that particular book. A ComboBox is an appropriate choice for author selection component in this view. You will need to populate the ComboBox with a list of authors (which you can pass into the detail view in your changeView method. Note that if the displayed book already has an author, you will need to make sure that author is selected initially when the drop-down list is initialized.

Also, be sure to implement an Audit Trail view for the Book Detail View. You can reuse the Audit Trail View component from your Author Detail View, or design a new way to show a Book’s audit trail.

8. Make sure your BookDetailView prompts the user to save if the user tries to transition to a different view AND

a. either the Book is new
b. or if viewing a previously inserted book and the GUI field data have changed

This is similar to the save prompting you implemented in the AuthorDetailView in Assignment 3.

9. To maintain referential integrity between your authors and their books, you must either:

a. add a Foreign Key Constraint to your database that prevents or cascading deletes books if the associated author is deleted
b. or modify your delete author method to first delete an author’s books if he/she is being deleted.

DELIVERABLES:

1. Make this assignment its own Eclipse Java project called Assignment4. In your main Java class, include a comment with “CS 4743 Assignment 4 by <your name>”.
2. Export the entire project to a zip file called assignment4.zip and submit the resulting zip file to Blackboard.

LATE POLICY:

-10 pts	few hours to 24 hours late
-20 pts	24 hours to 48 hours late
half credit	> 48 hours late

RUBRIC

30 pts	Optimistic locking implemented on Edit Author functionality
20 pts	Book list view and controller implemented correctly, including book deletion
10 pts	Book detail view uses a drop-down list (or similar intuitive metaphor) for selecting an Author
10 pts	Book creation implemented correctly
10 pts	Book updating implemented correctly
10 pts	Audit trail correctly implemented for books
10 pts	Book detail view prompts user to save if Book is new or existing book data have changed