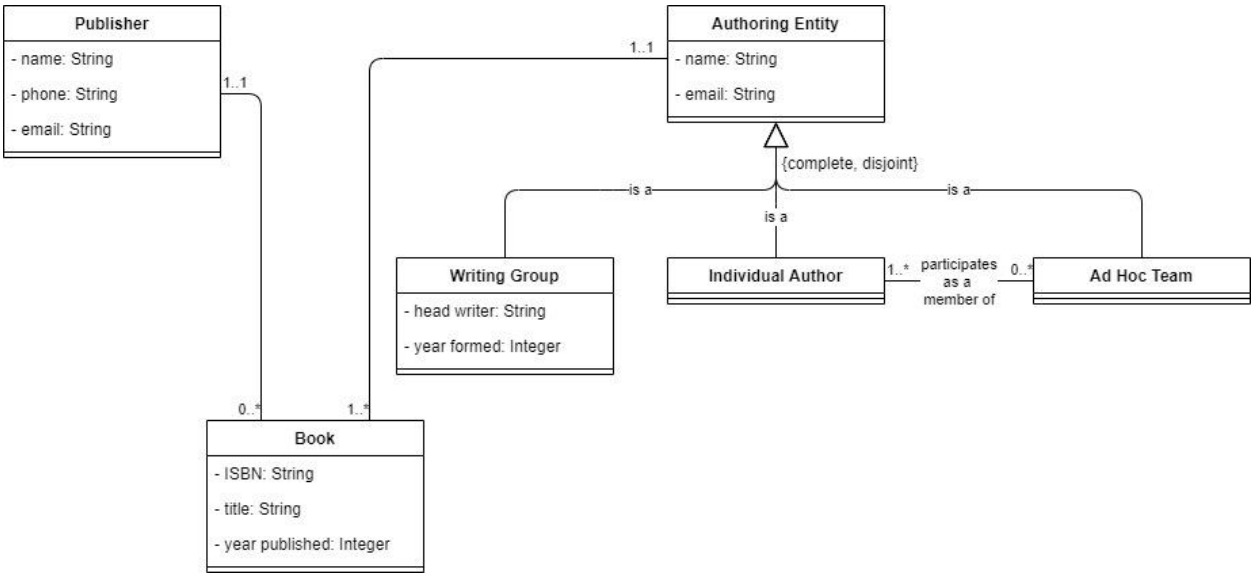
CECS 323 PROJECT: JPA - BOOKS

OBJECTIVE: Get an introduction to coding in Java Persistence API (JPA) using a small number of common JPA constructs.

INTRODUCTION: JPA has annotations for every mapping of the object model to the relational model. Many of those annotations are just variations on a theme, and therefore easy to pick up and intuitive once you have some experience with JPA. Others are more distinctive and take longer to master. But overall, we do not have anywhere near enough time to devote to JPA that we would need if we were going to get a broad understanding of everything that it does. Instead, this project will concentrate on just a few of the constructs in JPA as a demonstration of the overall process for mapping from the OO model to the relational.

Frequently, you will be asked to start with an existing database and either update an existing application that accesses that database, or write a new, OO application to access the existing database. Starting a project from scratch is rare in a mature environment. In this project, I will give you DDL to create the tables that you need here. **Do not modify the table structure in any way** in your project.

Your job is to write the Java entities (annotated POJOs) for all the classes in this model:



Functional Requirements

To prove your annotations, you will build a main application that will perform the following tasks:

1. Add new objects:
    a. Add a new Authoring Entity instance:
        i. Writing Group
        ii. Individual Author
        iii. Ad Hoc Team

iv. Add an author to an existing Writing Group
   b. Add a new Publisher
   c. Add a new Book
2. List all the information about a specific Object:
   a. Publisher
   b. Book (include the publisher and the Authoring Entity information)
   c. Writing Group
3. Delete a Book – be sure to prompt for all the elements of a candidate key. Make sure that the book actually exists.
4. Update a Book – Change the authoring entity for an existing book.
5. List the primary key of all the rows of:
   a. Publishers
   b. Books (show the title as well as the ISBN)
   c. Authoring entities – and supply the **type** of authoring entity for each one as well.

Error Handling:

There are two broad classes of errors that might crop up in your application: user input errors and database errors.

User input errors include such things as giving a menu item number that is out of range or entering a text string where you need a number. If your user interface does not protect the application from those sorts of errors, I will not take off points. I would expect that your personal sense of pride in doing a good job would prompt you to deal with that if you have the time.

However, database errors will cost you points. They are:

- Trying to insert an object that already exists. This comes down to checking the uniqueness constraint(s) on your tables.
- Trying to insert an object that references a non-existent object. For instance, trying to put an author into a non-existent writing group.
- Trying to delete an object that does not exist.
- Updates
  - Trying to update an object that does not exist
  - Updating an object such that it violates one or more constraints
    - Uniqueness
    - Foreign key

You can either prevent database errors or use Java try/catch blocks to trap them and try again. If you prevent database errors, there are two general strategies: guide the user input or test the user input. For example, you prompt the user for a book to delete. You provide them a numbered list of books in the database, and prompt them for a number to indicate which book they wish to delete. That

would be guided input. The other way to prevent errors would be to prompt the user to tell you which Book they want to delete, then perform a select on the database to see whether that book is present in the database. If it is not, tell the user and prompt them again. If it is, proceed to delete it.

Either way, you **must** provide the user with sensible error messages. By "sensible", I mean that they make sense to a businessperson. Just telling the user "you violated uniqueness constraint publishers_pk" is worse than useless. Instead, tell them "I am sorry, but there is already a publisher by that name in the database. Please try again." If your dialog with the user is cryptic, it only frustrates them.

Using Bind Variables

Whenever you write SQL for JPA to process, you have two basic approaches that you can take: create the query filled in as a single String value and execute it all at once or use bind variables. I will **require** that you use bind variables. You will find several examples of queries that use bind variables here. The bind variable can stand for any literal value that you might put into a SQL DML statement. You cannot use bind variables to stand in for things like table names or column names.

User Interface:

This project will already have quite a bit of technology that you have to master. I am not going to ask you to use a lush GUI framework to build this. Instead, just use the **console** for the input and output. That way, we can concentrate on the database portion of this, and leave the swanky user interface for later.

Coding Standards

While this is not a course in Java programming, I **am** going to hold you to several coding standards. These will help you in the long run. Following them from the start rather than try to rush to get your code to conform to these standards at the end is by far the wiser choice. Find the coding standards here.

PROCEDURE: I strongly suggest that you take this a chunk at a time, test copiously, and test frequently. Your sequence of events should be:

1. Install IntelliJ IDEA Ultimate and use it for the JPA lab. That will be a good way to test to make sure that you have a working development environment.
2. Do your first annotation. I suggest Publishers.
    a. Write the code to prompt the user for a new publisher.
    b. Then write the code to store a publisher.
    c. Test your annotations by adding and deleting publishers.
3. Perform the rest of the annotations in a similar process.
4. In your main program, develop a simple menu system that will allow you to find out from the user what they want to do.

5. Flesh out the rest of the functionality listed above under the heading of **Functional Requirements**.

WHAT TO TURN IN:

- The relation scheme diagram for the project.
- Your team's IntelliJ project. I want the whole thing, from the root of the project on down.
- The console output from a complete test run of your application.
    - Once in Derby
    - Once in MySQL
- Your team's issues log. You can find the template for those issues log here. There will be several decisions that your team needs to make along the way, and I want you to be organized and deliberate about how you document those decisions.
- Your team's assignment roster. You can find a template for that here.
    - The team as a whole will turn in the assignment roster **without** any assessments of the individual task output.
    - Each one of you will then fill out a separate copy of the teams assignment roster for each of the members of the team (including yourself) to assess how well everyone did on each of their assignments.
- We will take time during lab to demonstrate your application. Technically, that will not be anything tangible that you must turn in, but I will base part of your grade on your demonstration.

Note that I will provide a grading rubric for this assignment in BeachBoard so that you can see the point breakdown for the project.