# BookLibrary

## Documentation

The objectives were 1. BookLibrary

Serves as a library for storing collection of books. Choose the correct collection (consider

search performance). Following functionality needs to be supported:

- contains a collection of books and their quantity(the same book can be registered

multiple times - there can be more pieces of it in the library, right?)

- registering a new book

- retrieving a book from library (borrowing it)

- returning a book to library

2. Librarian

Represents your virtual assistant for working with the library (like the lady that works in the

library and you ask her for books). It needs to support communication with you through the

console, more specifically you can ask it to:

- list you the books present in the library (and their quantities)

- get you a book from the library you want to borrow

- put a book that you're returning back to the library

3. Book

Create a class for book that holds some information about the book, like name, value, ....

Use at least 5 properties. Provide read-only access to these information - they can't be

modified, once instance is created.

Demonstrate use of inheritance on the book class - create at least 2 subclasses for a special

kind of book with some extended functionality (f.e. book signed by author that has double

value compared to unsigned book).

Revitalization

White writing program in Java using PostgreSQL database

The program implements 9 classes and 3 subclasses(classes Book BookBorrower BookQuery BookRegistrar BookReturner LibraryBookRetrieval LibraryMenu librarySystem Main subclasses BookCommentator BookRater BookViewerAndRater

For the first task, we did the following

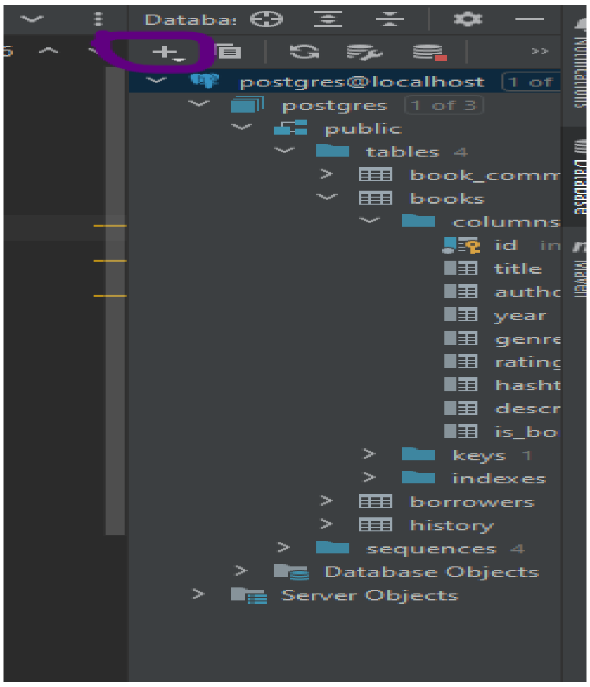
- contains a collection of books and their quantity(the same book can be registered

multiple times - there can be more pieces of it in the library, right?)

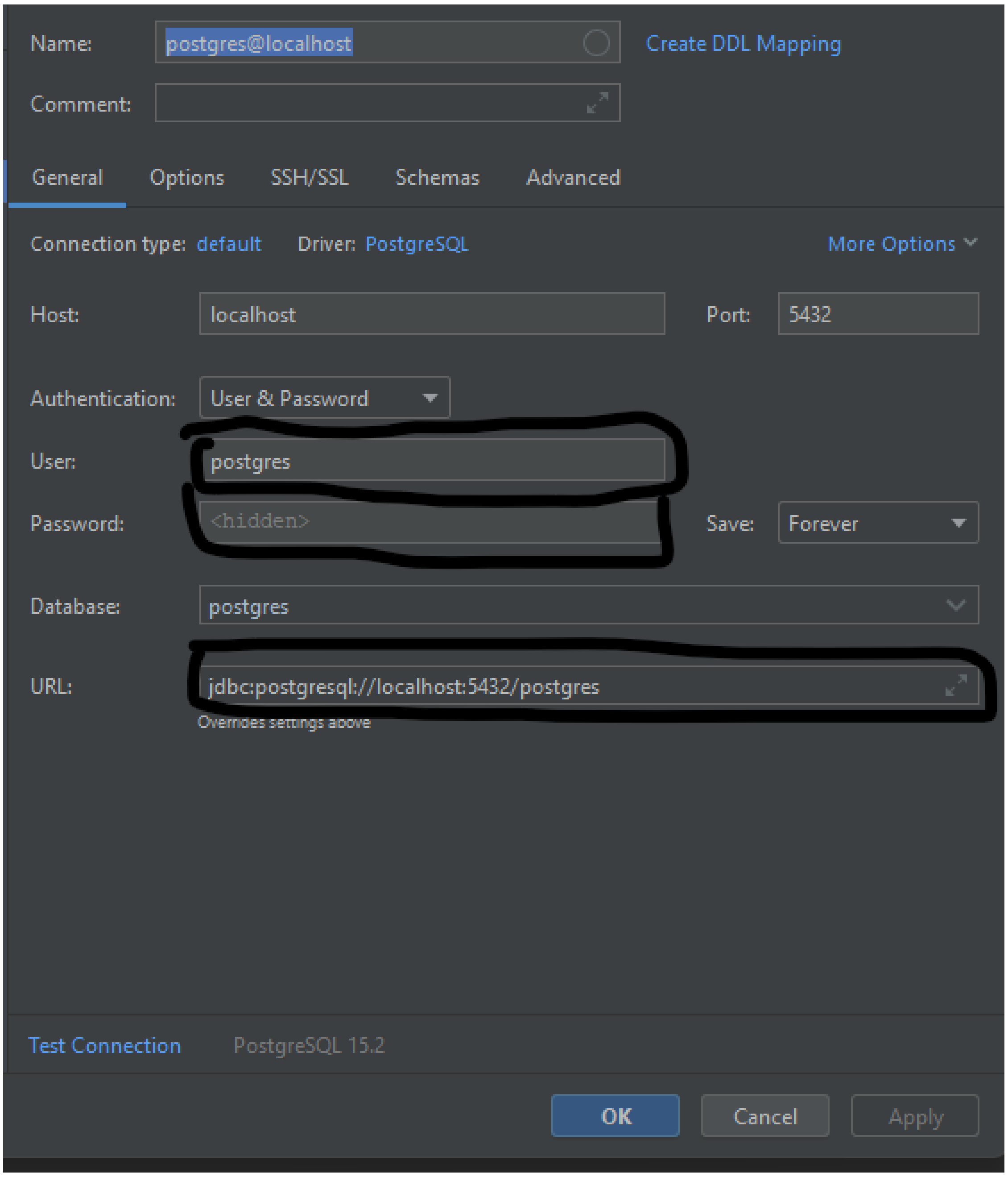
for this task we made we realized a database for storing and recording books

You need to tweak the database to make the code work

it's done in the following way downloaded from the website https://www.postgresql.org/ install and then in the program intellij idea click on the plus sign on the right and select the elephant. pictures shown



You will need to keep your login and password, which you used when installing Datibase.



Name: postgres@localhost

Comment:

Connection type: default Driver: PostgreSQL

Host: localhost Port: 5432

Authentication: User & Password

User: postgres

Password: <hidden> Save: Forever

Database: postgres

URL: jdbc:postgresql://localhost:5432/postgres

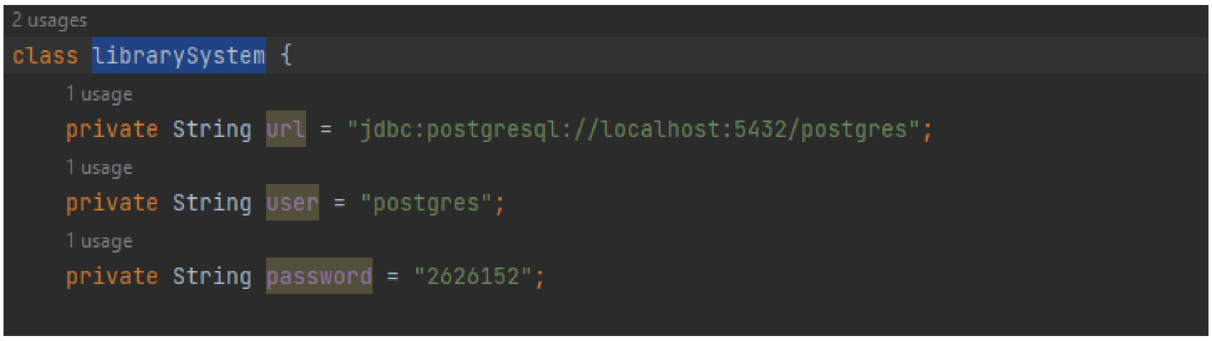
Overrides settings above

Test Connection PostgreSQL 15.2

and you also need to replace them in the code. BookQuery Book LibrarySystem

in these classes

for this task of writing a new book was the next solution



```
2 usages
class librarySystem {
    1 usage
    private String url = "jdbc:postgresql://localhost:5432/postgres";
    1 usage
    private String user = "postgres";
    1 usage
    private String password = "2626152";
```

## `create a` **`BookRegistrar`** `class`

Class Fields:

- `private Connection connection`: A field holding the database connection. The class takes the connection through the constructor, making it independent of how the connection is obtained.

Constructor:

- `public BookRegistrar(Connection connection)`: A constructor taking a database connection as a parameter and storing it in the class field.

Method `registerBook()`:

- It uses a `Scanner` for user input of book information.
- After input, the method constructs an SQL query to insert a new book record into the database.
- A `PreparedStatement` is created to execute the query with parameterized values to prevent SQL injection attacks.
- Values are inserted into the query using methods like `setString`, `setInt`, etc.
- The query to add the book record to the database is executed.

- Upon successful execution, a message about the successful book registration is displayed. In case of an error, an error message with details is shown.

Error Handling:

- In case of a `SQLException`, an error message about the book registration failure is displayed along with error details.

Note: The code assumes the existence of a database with a `books` table having fields such as `title`, `author`, `year`, `genre`, `rating`, `hashtags`, and `description` to store information about books.

`retrieving a book from library (borrowing it)`

`create a BookBorrower`

## `class BookBorrower`

Class Fields:

- `private Connection connection`: A field holding the database connection. Similar to the `BookRegistrar` class, this class takes a connection through the constructor.

Constructor:

- `public BookBorrower(Connection connection)`: A constructor that accepts a database connection and stores it in the class field.

Method `borrowBook()`:

- Uses a `Scanner` for user input.
- Collects the borrower's name and the book ID they wish to borrow.
- Checks if the entered data includes a valid integer as the book ID; otherwise, it prints an error message and returns.
- Executes a SQL query to check whether the specified book is already borrowed by someone else.

        - If the book is already borrowed, it prints a message indicating that the book is not available for borrowing.
        - If the book is not found in the database, it prints a message stating that the book with the specified ID was not found.
    - If the book is available for borrowing, it updates the database to mark the book as borrowed.
    - Prints a success message if the book is successfully borrowed.
    - In case of any SQL exceptions during the borrowing process, it prints an error message with details.

Note: This code assumes that there is a `books` table in the database with a field named `is_borrowed` to track the book's availability status and a unique identifier for each book (`id`). Additionally, the code includes a placeholder for recording borrower information and history, which can be expanded upon based on specific requirements.

```
- returning a book to library
 create a BookReturner
class BookReturner
```

- `private Connection connection`: A field holding the database connection. Similar to the previous classes, this class takes a connection through the constructor.

Constructor:

- `public BookReturner(Connection connection)`: A constructor that accepts a database connection and stores it in the class field.

Method `returnBook()`:

- Uses a `Scanner` for user input.
- Consumes a newline character to handle potential issues with reading input after numerical input.
- Collects the borrower's name and the book ID they wish to return.
- Checks if the entered data includes a valid integer as the book ID; otherwise, it prints an error message and returns.
- Executes a SQL query to check if the specified book is borrowed by the specified borrower.
    - If the book is not borrowed by the specified borrower, it prints a message indicating that the book was not borrowed by the specified person.
- Updates the book status in the database to mark it as "not borrowed."
- Removes the borrower record associated with the returned book.
- Records the return information in a history table.
- In case of any SQL exceptions during the return process, it prints an error message with details.

Note: This code assumes the existence of a `books` table with an `is_borrowed` field, a `borrowers` table to store borrower information, and a `history` table to track book borrowing and return history. Adjustments may be needed based on the specific database schema and requirements.

Represents your virtual assistant for working with the library (like the lady that works in the

library and you ask her for books). It needs to support communication with you through the console, more specifically you can ask it to:

- list you the books present in the library (and their quantities)

- get you a book from the library you want to borrow

- put a book that you’re returning back to the library

This is real in this class **`LibraryMenu`**

Fields:

- The class has a static `Connection` field named `connection` to hold the database connection.

Method `searchBooks()`:

- Calls a method from the `BookQuery` class to search for books. The method in `BookQuery` is not shown in the provided code snippet.

Constructor:

- The constructor takes a `Connection` object as a parameter and initializes the `connection` field.

Method `displayMenu()`:

- Simulates a narrative journey to the library, presenting various scenes.
- Prompts the user with a menu to perform different actions related to borrowing, returning, searching, viewing information, and leaving comments on books.
- Uses a loop to repeatedly prompt the user for input until the user chooses to exit (`choice == 8`).
- Depending on the user's choice, it invokes methods from other classes (e.g., `BookBorrower`, `BookRegistrar`, `BookViewerAndRater`, etc.) to handle specific actions.

Static Method `handleSubMenu(Scanner scanner)`:

- Displays a submenu related to book borrowing.
- Allows the user to view the list of books, borrow a specific book, or return to the main menu.

Private Method `handleSubMenu3(Scanner scanner)`:

- Displays a submenu related to viewing book information and comments.
- Allows the user to view information about a specific book or view comments on a book.

Private Method `handleSubMenu4(Scanner scanner)`:

- Displays a submenu related to leaving comments or rating a book.
- Allows the user to rate a book or leave a comment.

Private Method `displayIntroductoryMessage()`:

- Displays an introductory message for the library simulator.

Private Method `clearConsole()`:

- Clears the console by printing multiple empty lines.

Private Method `isUserReady()`:

- Asks the user if they are ready to start the "game" (library simulation).

Overall, the `LibraryMenu` class seems to serve as the main interface for users to interact with the library system. It uses a narrative approach to create an engaging user experience and offers a variety of options for users to explore the functionalities of the library.

This is real in this class **`BookQuery`**

Fields:

- The class has static fields `url`, `user`, and `password` to store the connection details for the PostgreSQL database.

Method `BookQuery(String[] args)`:

- Provides a user interface for searching books based on various criteria.
- Uses nested loops to continuously prompt the user for search criteria until they choose to exit.
- For each search criterion, it prompts the user for input and invokes methods to handle different types of search criteria (e.g., genre, author, rating, year, hashtag).

Private Methods `getBooksByGenre`, `getBooksByAuthor`, `getBooksByRating`, `getBooksByYear`, `getBooksByHashtag`:

- These methods take a specific search criterion and value, then query the database to retrieve a list of books that match the given criterion.

Private Method `getBooks(String columnName, Object value)`:

- A generic method that retrieves books based on a specified column name and value from the database.

Private Method `displayBooks(List<Book> books, String message)`:

- Displays the list of books retrieved from the database based on the search criteria.
- Prints book details such as ID and title.

Inner Class `Book`:

- Represents a simple data structure for storing book information (ID and title).

Note:

- The code appears to have some commented-out sections and parts that reference methods like `handleSubMenu` from the `LibraryMenu` class. The usage and purpose of these parts are not clear from the provided code snippet.

Recommendation:

- It's good practice to handle and log exceptions more appropriately, especially in a production environment, to provide better error handling and debugging information.

3. Book

Create a class for book that holds some information about the book, like name, value, ....

Use at least 5 properties. Provide read-only access to these information - they can't be

modified, once instance is created.

Demonstrate use of inheritance on the book class - create at least 2 subclasses for a special

kind of book with some extended functionality (f.e. book signed by author that has double

value compared to unsigned book).

This task was the hardest for me because we didn't know what to do with subclasses. was implemented so that the class Book it outputs information about books

**`Book`**

Database Connection:

- The URL, username, and password for connecting to the PostgreSQL database are set.
- Methods `getUrl()`, `getUser()`, and `getPassword()`

provide access to this connection information.

Method `viewBookInfoById(int bookId)`:

- Takes a book identifier (`bookId`) as input.
- Constructs an SQL query to select information about the book with the specified identifier.
- Sets the parameter in the prepared statement.
- Establishes a connection to the database, executes the query, and retrieves the results.
- If results are found, it prints information about the book (ID, title, author, year, genre, rating, hashtags, description). Otherwise, it prints a message that the book with the specified ID is not found.
- Handles potential exceptions during the query execution.

and its subclasses BookCommentator BookRater BookViewerAndRater

**BookCommentator**

Constructor:

- The `BookCommentator` class has a constructor that takes the `url`, `user`, and `password` as parameters and calls the superclass (`Book`) constructor using `super(url, user, password)`. This constructor is used to initialize the connection details inherited from the `Book` class.

Method `addCommentToBook(int bookId, String commenterName, String commentText)`:

- Adds a comment to a book in the database.
- Constructs an SQL query to insert a new comment into the `book_comments` table with the provided book ID, commenter name, and comment text.
- Establishes a connection to the database, prepares a statement, sets the parameters, and executes the update.

- Prints a success message if the comment is added successfully; otherwise, it prints an error message if an exception occurs during the execution.

Method `createCommentsTable()`:

- Creates the `book_comments` table in the database if it does not exist.
- Establishes a connection to the database, creates a statement, and executes the SQL query to create the table.
- Prints a success message if the table is created successfully; otherwise, it prints an error message if an exception occurs during the execution.

Note: It seems that there's a potential issue with the `url` field in the `BookCommentator` class. The `url` field in `Book` is static, but in `BookCommentator`, it's an instance variable.

## `BookRater`

Constructor:

- The `BookRater` class has a constructor that takes the `url`, `user`, and `password` as parameters and calls the superclass (`Book`) constructor using `super(url, user, password)`. This constructor is used to initialize the connection details inherited from the `Book` class.

Method `setBookRating()`:

- Prompts the user to enter the ID of the book and the new rating (between 1 and 10).
- Constructs an SQL update query to set the new rating for the specified book ID.
- Establishes a connection to the database, prepares a statement, sets the parameters, and executes the update.
- Prints a success message if the rating is updated

successfully; otherwise, it prints a message if the book with the specified ID is not found or if an exception occurs during the execution.

Note: Similar to the previous comments, it's recommended to use the `getUrl()`, `getUser()`, and `getPassword()` methods from the superclass (`Book`) to retrieve the connection details consistently, especially since these fields are static in the `Book` class.

## **`BookViewerAndRater`**

Constructor:

- The `BookViewerAndRater` class has a constructor that takes the `url`, `user`, and `password` as parameters and calls the superclass (`Book`) constructor using `super(url, user, password)`. This constructor is used to initialize the connection details inherited from the `Book` class.

Method `viewBookComments(int bookId)`:

- Takes a book ID as input.
- Constructs an SQL query to select comments for the specified book ID from the `book_comments` table.
- Establishes a connection to the database, prepares a statement, sets the parameter, and executes the query.
- Prints the details of each comment, including comment ID, book ID, commenter name, and comment text.
- Handles potential exceptions during the query execution.

Method `setBookRating(int bookId, double newRating)`:

- Takes a book ID and a new rating as input.
- Constructs an SQL update query to set the new rating for the specified book ID in the `books` table.
- Establishes a connection to the database, prepares a statement, sets the parameters, and executes the update.
- Prints a success message if the rating is updated successfully; otherwise, it prints a message if the book with the specified ID is not found or if an exception occurs during the execution.

Note: Similar to the previous comments, it's recommended to use the `getUrl()`, `getUser()`, and `getPassword()` methods from the superclass (`Book`) to retrieve the connection details consistently, especially since these fields are static in the `Book` class.

We also created such classes as `librarySystem and LibraryBookRetrieval`

**`librarySystem`**

Fields:

The class has private fields url, user, and password representing the connection details for a PostgreSQL database.

Method run():

Attempts to establish a connection to the PostgreSQL database using the provided connection details (url, user, password).

If the connection is successful, it prints a message indicating the successful connection and proceeds to create an instance of the LibraryMenu class.

Calls the displayMenu() method of the LibraryMenu instance, which appears to be responsible for managing the main menu of the library system.

If the connection fails, it prints a message indicating the failure.

Exception Handling:

Catches SQLException and prints the associated error message.

Catches InterruptedException and rethrows it as a RuntimeException.

Recommendations:

It's good practice to close the database connection (conn) in a finally block to ensure that it is closed regardless of whether an exception occurs or not.

Consider adding more robust error handling, logging, and potentially displaying more meaningful error messages to users.

Overall, this class represents the starting point of the library system, attempting to connect to the database and launching the main menu through the LibraryMenu class.

The class **LibraryBookRetrieval**

Constants:

- Defines constant fields for JDBC connection details (JDBC_URL, USERNAME, PASSWORD).

Method LibraryBookRetrieval(String[] args):

- Initiates by setting a specific bookId for demonstration purposes.
- Calls the retrieveLibraryBookById method to retrieve information about a specific book using the provided bookId.
- Prints the details of the retrieved book if it exists, otherwise, it prints "Book not found."
- Calls the viewAllBooks method to display information about all books in the library.

Method retrieveLibraryBookById(int bookId):

- Establishes a database connection using the constant connection details.
- Defines an SQL query to select all columns from the "books" table where the book's ID matches the provided bookId.
- Uses a PreparedStatement to execute the query and retrieve the results as a ResultSet.

Method viewAllBooks():

- Establishes a database connection using the constant connection details.
- Defines an SQL query to select all columns from the "books" table, ordered by the book's ID.
- Uses a PreparedStatement to execute the query and retrieves the results as a ResultSet.
- Prints the details of each book in the result set.

Method closeResultSet(ResultSet resultSet):

- Closes the provided ResultSet if it is not null.

Exception Handling:

- Catches SQLException in various places and prints the stack trace.

Note:

- The LibraryBookRetrieval class name suggests it might be intended for initialization, but it's used as a regular method. Consider changing the method name or the class name for clarity.

Overall, this class provides methods for retrieving information about a specific book and displaying details for all books in the library from a PostgreSQL database.