Assessment 3: Assignment 2

William Hall

Task Outline

This task is to implement a Library Management System(LMS) which upholds the three principles of Data Oriented Programming(DOP). These are: Code is separated from data; Data is immutable and Data access is flexible. The task has 7 Requirements which need to be met: Two kinds of users(Librarians and Members), Users can log in via email and password, Members can borrow books, Members and librarians can search for books by title or author, Librarians can block and unblock members, Librarians can list the books currently on loan to a member and there can be several copies of a book.

Implementation

To achieve DOP all code classes are implemented with static methods only, therefore within my code each class is a helper class. Additionally, Data classes must only have members and no methods. This is implemented with @Value from the Lombok.

All Data classes follow a similar structure:

```
@Value

Class Data{

        DataMember DataMemberName;

        DataMember DataMemberName2;

}
```

This can have as many DataMembers as defined by the user. The Benefit of using @Value from the Lombok package is that this makes data immutable and automatically creates boilerplate code for the DataMembers. This makes the code more readable and makes less work for the programmer.

For the Code classes below, the get methods are provided as the boilerplate code for the Data classes by @Value

BookItemCode

This is the code relating to the BookItemData class.

Methods:

```
addBookItem(BookItemData BookItem){
    for(Book in Library){
        if(Book == BookItem.getName()){
            Library.getLibCatalog().getListOfBooks().getBookItems().add(BookItem)
        }
    }
}
```

Complexity: O(n), where n is the amount of BookItems in the Library.

BookLendingCode

We find the book and the member and add the lendingData to each of them

Methods:

```
addBookLendingData(BookItemData Book, MemberData Mem){
    var LentItem = new BookLendingData(Book.getName(), Mem.getEmail(),
    Book.getUniqueBookId);
    for(Book in Library){
        if(Book == BookItem.getName()){
    Library.getLibCatalog().getListOfBooks().getBookItems().getBookLentItemsList().add(B
ookLentItem)
        }
    }
    for(Member in Library){
        if(Member.getName() == Mem.getName()){
            Library.getMembers().getLentItems().add(BookLentItem)
        }
    }
}
```

Complexity: O(n+k), where n is the bookitems and k is the members

<u>CataLogCode</u>

This is the code for the catalogue of books in the library.

Methods:

```
CatalogCodeIni(List<BookData> Lis){ //Creates a new Catalog
        Library = new LibraryData(Library.getLibrarians(), new CatalogData(Lis),
Library.getMembers);
}
```

Complexity: O(1)

```
updateOneCatalog(BookData Book){ //adds one book to existing catalog
        Library.getCatalog().getBooks().add(Book);
}
```

Complexity: O(1)

```
updateManyCatalog(List<BookData> Lis){
        Library.getCatalog().getBooks().addAll(Lis);
}
```

Complexity: O(1)

<u>LibrarianCode</u>

The code class for Librarians

```
addLibrarian(String email, String Password){ //This can be done with LibrarianData Also.
        Library.getLibrarians().add(new LibrarianData(email, password, Boolean.FALSE))
}
```

Complexity: O(1)

```
blockMember(String MemEmail){ // unblock method is very similar
```

```
        for(Member in Library){
                if(Member.getName() == MemEmail){
                        Library.getMembers().remove(Mem);
                        Library.getMembers().add(new MemberData(Mem.getEmail(),
Mem.getPassword, Mem.getLoggedIn, Boolean.TRUE))// For Blocked
                }
        }
}
```

Complexity: O(n), where n is the members in the library

```
ListBooks(String MemEmail){
        for(Member in Library){
                if(Member.getName() == MemEmail){
                        Library.getMembers().remove(Mem);
                        Library.getMembers().add(new MemberData(Mem.getEmail(),
Mem.getPassword, Mem.getLoggedIn, Boolean.TRUE))// For Blocked
                }
        }
}
```

Complexity: O(n), where n is the members in the library

<u>MemberCode</u>

The code class for MemberData

Methods:

```
addMember(String email, String password){ // MemberData version is similar
        Library.getMembers().add(new MemberData(email, password, Boolean.False,
Boolean.False));
}

Borrow(String MemEmail, BookItemData BookItem){
        Var foundMember = Library.getMembers.find(MemEmail);
        If(foundMember = null){
                Return;
        }

        If(foundMember.getBlocked() == False){
                return
        }
        Var book = Library.getCatalog().GetBookList().find(Bookitem.getName());
        Book.remove(Bookitem);
        Book.add(new Bookitem(Bookitem.getName(), Mem.email, Bookitem.getUniqueId));

}
```

Complexity: O(n), where n is the number of Members in the library

<u>UserCode</u>

The code class which is a parent to Members and Librarian Code classes.

Methods:

```
Login(String email, String password){
        For(var Lib in Library.getLibrarians()){
                If(Lib.getEmail() == email && Lib.getpassword == password){
                        Library.getLibrarians().remove(Lib);
                        Library.getLibrarians.add(new LibrarianData(email, password,
Boolean.TRUE));
                        Return;
                }
        }
        For(var Mem in Library.getMembers()){
                If(Mem.getEmail() == email && Mem.getpassword == password){
                    Library.getMembers().remove(Lib);
                    Library.getMembers.add(new MemberData(email, password,
Boolean.TRUE, Boolean.False));
                    Return;
            }
      }
      System.out.println("Wrong Email or Password");
}
```

Complexity: O(n+j), where n is the Librarians and J is the Members

```
Search(String AuthTitle){
      String Output = "";
      For(Book in BookList){
            If(Book.getName() == AuthTitle){
                  Output + Book;
            }
            If(Book.Author.getName() == AuthTitle){
                  Output + Book;
            }
      }
      System.out.println(Output);
}
```

Complexity: O(n), where n is the number of books.

Discussion/Possible Improvements

Recreating the structures each time a change is required is very resource intensive as it is a much larger operation then just changing a single variable.

Implementing the Lists in the data structure as an Array List would reduce the implicit casting then changing lists.

As all classes have been implemented as helper classes, due to the restriction of methods needing to be static. This has made it difficult to implement the Member and Librarian Code classes as a User/Member/Librarian helper class can do everything required but has no strong link to the MemberData/LibrarianData classes. Using a wrapper could reduce this problem although a constructor cannot be static. Therefore, this would not abide by static only methods. A further issue with this is you cannot link actions to users who have performed the action. E.g., if a Librarian blocks a member you are unable to tell which librarian has done this.

This implementation relies heavily on a public static variable ‘Library’ being shared among all code classes. This is not ideal as anyone can access and change this at any time. This could be a problem in a real Library Management System it would be implemented as a distributed system, and these are prone to data races and could cause errors. A solution to this would be to implement strong lock-handling code.