**Software Design Document For Book API**

**Rishab Kumar (40199196)**

**Ravleen Kaur (40221236)**

**14-11-2022**

Table of Contents

1. Introduction
2. Technology Specifications
3. Use Cases
4. Class Diagram
5. Database Diagram
6. Junit testing
7. Refactoring Techniques

**Project Report**

# 1 Introduction

This is the project report for the Application OpenLibraryAI Driver which acts as a local independent shard of the bigger system. Data about the available books is fetched from the OpenLibrary’s central database repository through API and then inserted into the local version of the database. Users registered with the local system are then able to buy books and see which books are most popular in their local environment.

## 

**2. Technology Specifications**

|  |  |  |
| --- | --- | --- |
| 1. | IDE (Integrated Development Environment) | Eclipse |
| 2. | Version Control System | GitHub |
| 3. | Database | SQLite 3.36.0 |
| 4. | API | https://openlibrary.org/developers/api |
| 5. | Programming Language | Java 11 |
| 6. | Database API | Java Database Connectivity (JDBC) 4.3 |
| 7. | Project GitHub Link | https://github.com/Rishi-K/APP\_40199196\_40221236 |

## 

# 3 Use Cases

**Use-Case Model**

## 3.1 Actors

The User of this application is a generic user who wants to view the top books and authors available and purcharse books.

### 

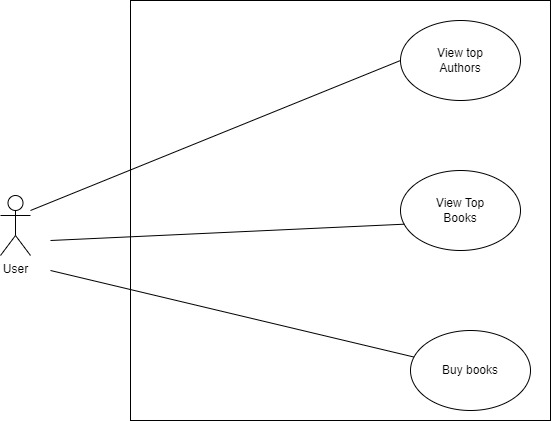
## 3.2 List of Use Cases

1. View Top books

2. View Top Authors

3. Purchase books

## 3.3 Use Case Diagrams



### .

**3 Design Overview**

## 3.1 Introduction

This is a console application which uses a set of data model classes whose attributes mimic the columns of the corresponding table. Thus there is a class User for representing a user which accepts data from the User table of the Database. Similarly Authors and Books class objects accept data from rows of Authors and Book tables respectively.

The Controller layer of Application has a Controller interface inherited by UserController, AuthorsController and BookController classes who perform the various CRUD operations using the User/Authors/Books object as data source for current operation.

The Controller classes use JDBC driver to connect to the database hosted on the sqlite3 database system.

APIReader class has functions to read the data on OpenLibrary servers using the Java.net libraries to request data through API url.

## 3.2 System Interfaces

### 

1 User Interfaces

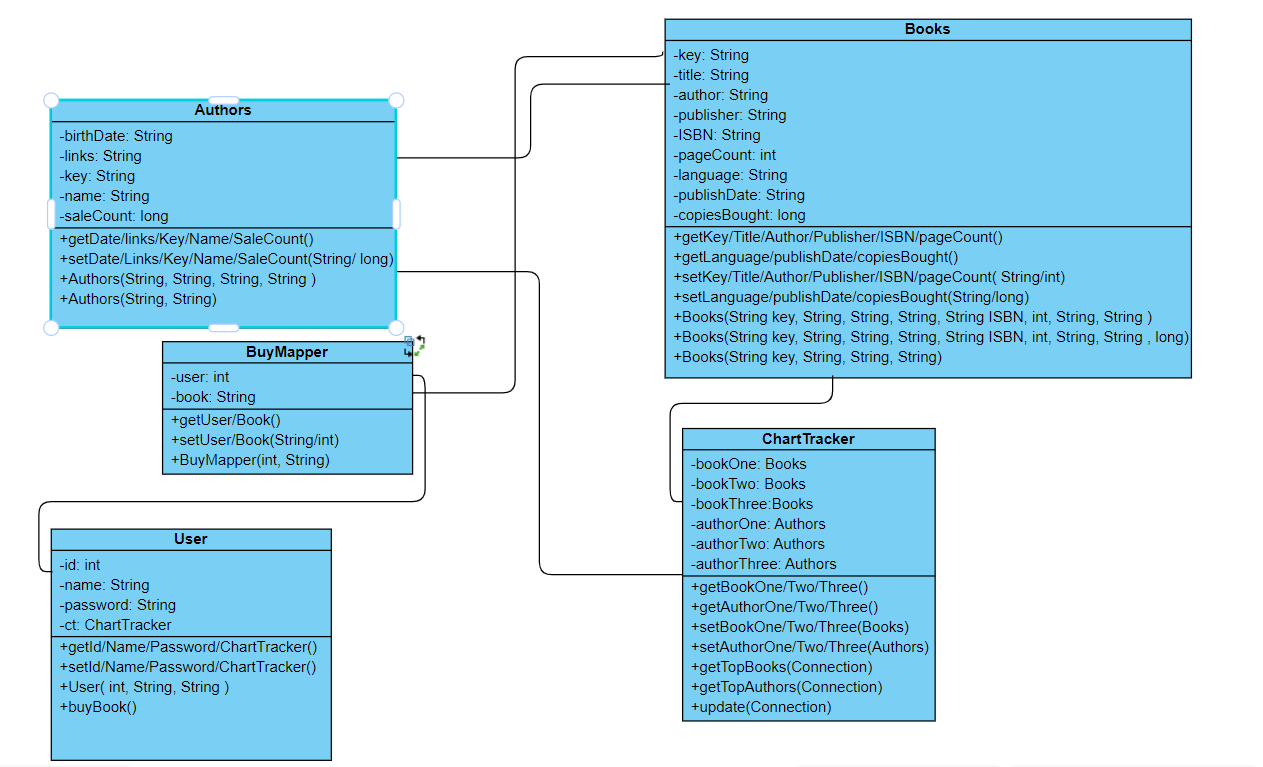
* The user interface for the system is a console which will allow the user to easily see the list of top authors and top books and purchase books.

2 Software Interfaces

* The software will need to interface with a database system SQLite to pull data from it and push data updates to it. The connection will be a standard database connection using JDBC or ODBC.

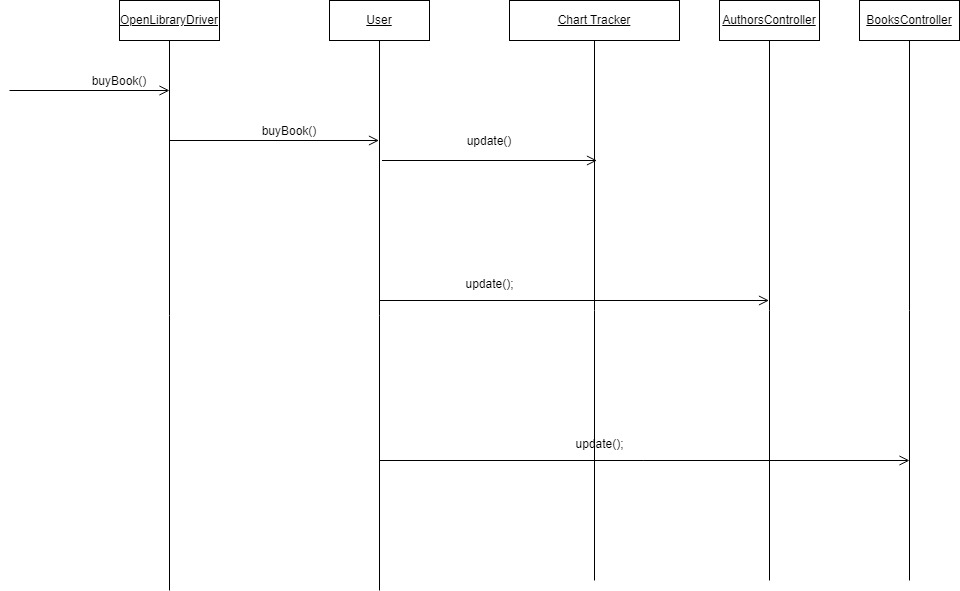
# 

**4. Class Diagram**



The above diagram represents the data model layer of the application which mimic the structure of the database table. Authors, Books, User and BuyMapper class attributes correspond to the columns of the Authors, Books, User and BuyMapper table so that the data can be exchnaged from the class objects to table rows and vice versa.

Apart from this, User class also has a ChartTracker object as it’s attributes. This allows the ChartTracker object to observe the state changes trigerred by the User’s buyBook function.



The classes in this layer make use of the **Observer design pattern**. A function call of buyBook by the User object triggers state changes for the Books object and the Authors object which then triggers corresponding changes in the Books and Authors table. Once User has notified the corresponding Controller classes to trigger the update of the corresponding tables, the state change of anothr Observer, ChartTracker object is triggered by the Subject i.e User by executing ChartTracker’s update function leading to change in the state of the ChartTracker object i.e the top Books and Top Authors data inside it.

APIReader class also acts as a factory for creating objects of Authors, Books as needed.

A picture containing table

Description automatically generated

The Controller classes ( displayed in above diagram ) which comprise the Controller layer of the application, follow an inheritance structure where the create delete update and read operations are inherited and implemented by the classes to connect to the SQLite database to perform these CRUD operations using JDBC. These Controller classes manage the flow of information to and from the database and the application.

Graphical user interface, application, Word

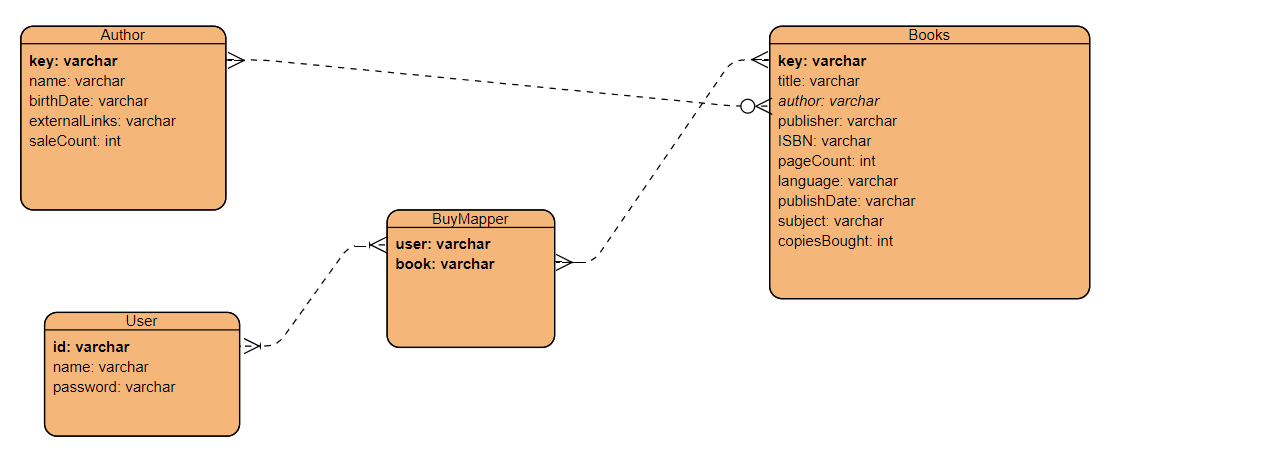
Description automatically generated

Since our application is a console application, OPenLibraryDriver class with the main method acts as the driver of our user interface displaying and redisplaying the Top Books and Top author chart, giving user option to LogIn, Start a Book Buying session.

APIReader class is a standalone class used to encapsulate the code for calling the OpenLibrary API based on different parameters and then read the json data, convert it into corresponding Authors or Books object

Firstly the User id and password is used to authenticate the user in application throguh openlibrary driver.The Data is being read from the API throught the API reader and open Library Driver which is then stored in the database through the controller classes. The chart Tracker serves as an Observer and whenever the book is purcharsed by the User class an update is triggered to Both Authors and ChartTracker which update the sale count and TopAuthors and Topbooks list respectively upon receiving the update message from User. The Results are then displayed by the openLibraryDriver class.

**5.Database diagram** :

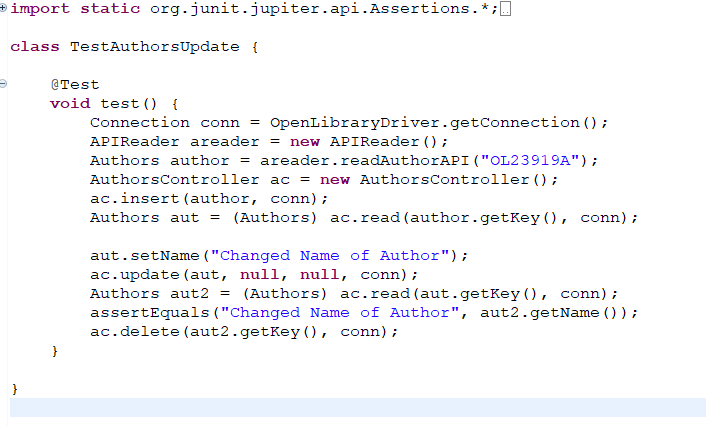


## There are 4 tables made using the SQLite database name Author which stores the information about the author and has a foreign key relationship with books. The data about the Authors is fetched from the API and stored in the author table.

Similarly the data is fetched for the User and books table and stored in it. Buy Mapper enables the application to store the information about the books bought by the user.

# 6. Junit testing tool

Unit test cases have been written to test the functionalities of Author Update, Book Update, User Update and Insert, delete and read operations for Users, Books and Authors.



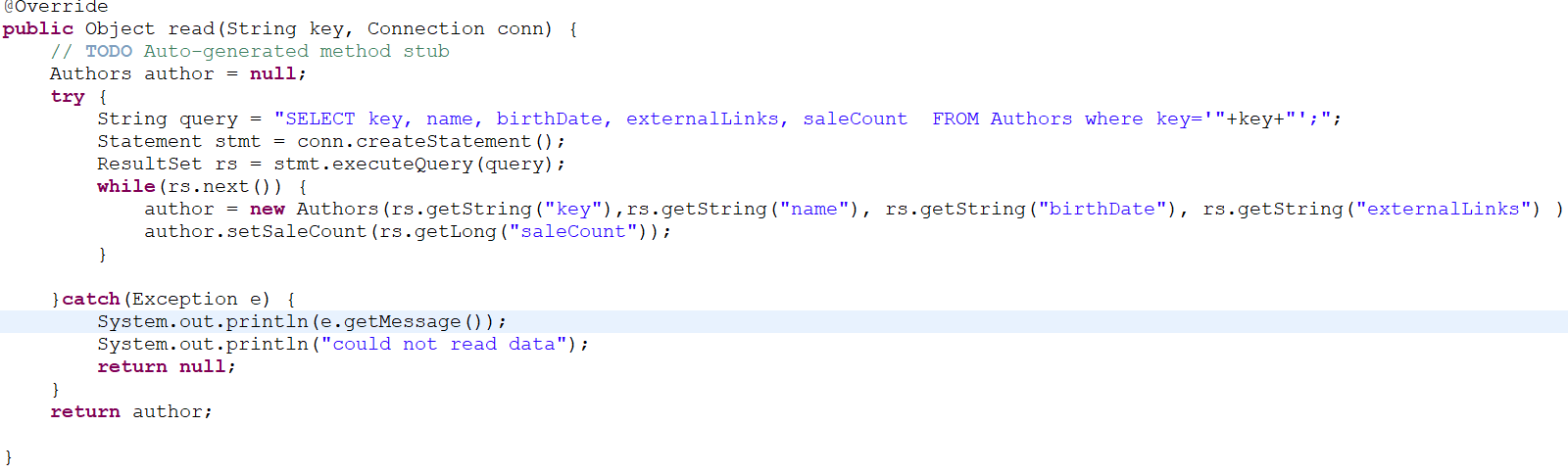
One of the examples of test cases implemented has been given above. The above codes changes the author name and tests the success of the code to check whether the update was successful or not.

7.Refactoring Techniques

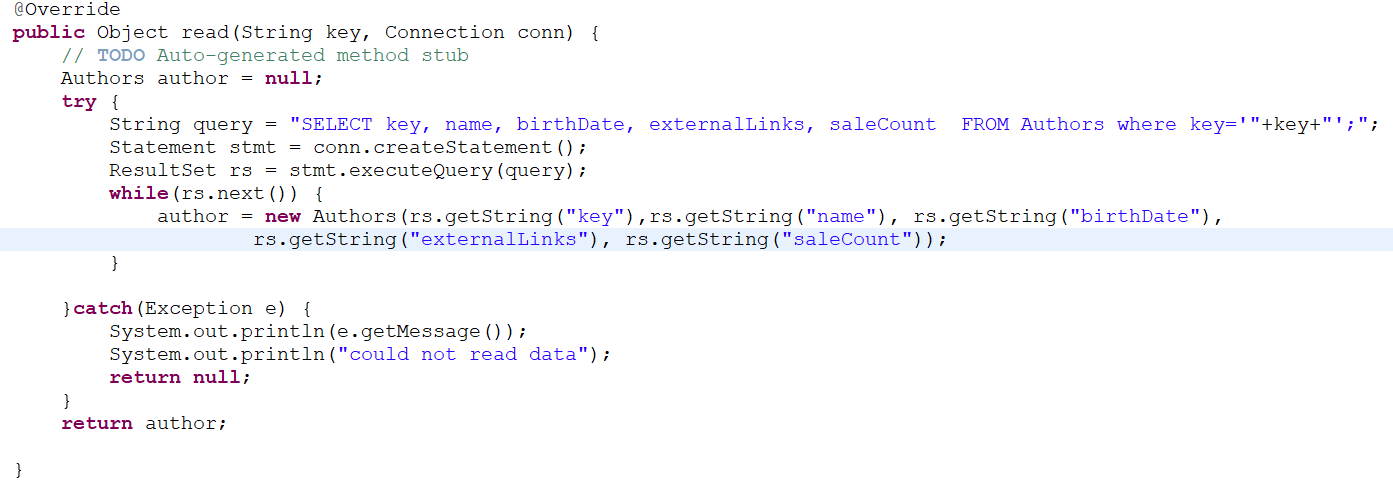
1. Replacing the call to method with overloaded constructor call.

There is already an overloaded constructor which accents salecount as a parameter present in the Author class. So replacing the code with the appropriate constructor call code.

Before Refactoring:



After Refactoring:



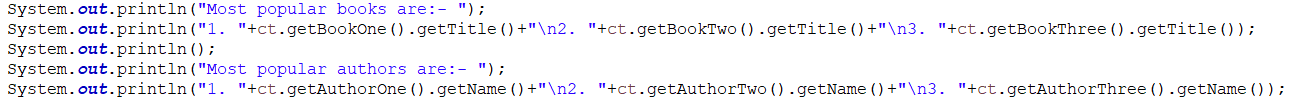
1. Extract Method

In the OpenLibraryDriver Class we extracted a section of code which performed a cohesive task of getting the list of Authors from the database

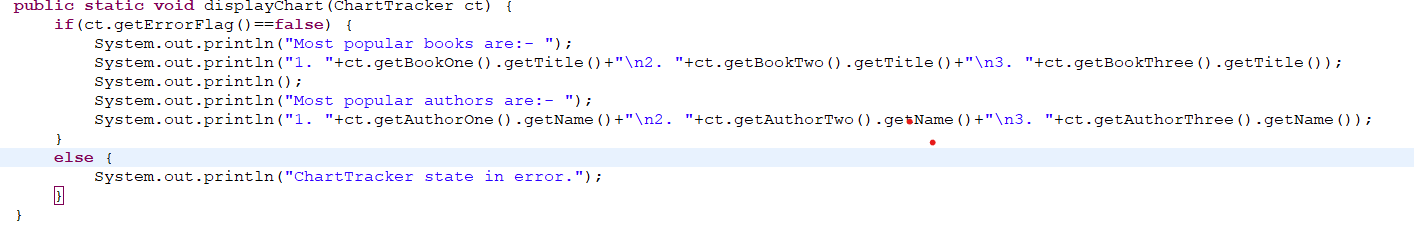


Similarly, we had code to generate topAuthors and TopBooks charts which we extracted into a method displayCharts().

Before Refactoring:



After Refactoring



1. New Class

A separate class ChartTracker was made to getTopbooks and getTopAuthors which were being called together multiple times to get the data. We Added required variables to a new class ChartTracker and packaging two methods into one single method call inside ChartTracker.

