Book Advisor

A system for tracking books you own, and to recommend the next one you should read

Tyler Lemieux

Assignment 1

5/5/20

For this assignment, you will describe and implement release 1 of your term project. You will incorporate *an abstract class*, *inheritance*, *upcasting or downcasting*, and *polymorphism*. You are free to choose a project that interests you but if you prefer, the instructor and your facilitator will be happy to suggest a topic. If you are already an experienced developer, this is an opportunity to build a challenging application (check with your facilitator if it requires significant API's) or discuss research with the instructor. It is fine to name a project with much more scope than you can hope to accomplish in the course (as in the example above): we will not hold you to completing everything associated with it. What we do expect is that you specify and implement a set of do-able requirements within such scope.

Submit this completed Word document, replacing all parts "Replace this …", observing and retaining the gray text. Your materials —in black 12-point Times New Roman—should not exceed 5 pages excluding references, figures, and appendices. Use the Appendix sections for additional material if you need to. These will be read only on an as-needed basis.

We want you to develop in Eclipse preferably or else IntelliJ (talk to your facilitator about exceptions). As you code, use JUnit tests whenever possible but certainly by week 2—package-by-package, class-by-class, and method-by-method, except for trivial methods and those requiring I/O. Use non-Junit classes for testing the latter. Keep the evaluation criteria in mind, listed at the end.

For this assignment, you do not need to read data from a file—you can build all data into the code.

Include a ReadMe file describing where to run the application from, and including necessary execution notes. All JUnit tests will be assumed runnable.

1.1 SUMMARY DESCRIPTION

One- or two-paragraph overall description of your proposed term project—half-page (12-point Times New Roman) limit. By the end, term projects will incorporate most of the techniques discussed in the course. To do this, you may need to alter the direction of your project or introduce an additional project. You may alter this or even replace it as the semester progresses. You will probably find it useful to use your project acronym.

This project will be a system that will allow a user to enter books they have and get recommendations on what they should read next. To use the application, called Book Advisor, the user will be able to enter books they with some information such as genre, some keywords from the description, and title. As they complete books they will be able to tell the application they completed it and receive recommendations based on keywords for books they may want to read next.

1.2 PROJECTED I/O EXAMPLE FROM PROJECTED COMPLETED PROJECT

Provide an example of projected *concrete* output for designated input. You will not be held to fulfilling exactly this—it is just explanatory at this point, to indicate where your project is going. We recognize that project direction and details will change as the term progress. This section refers to the project as a whole, not just to what you will produce this week, so we can gain an idea of what you have in mind overall.

Here is an example of a simple Book Advisor test.

Book Advisor:
Books you own:

1. Title: Harry Potter and the Chamber of Secrets | Genre: Fantasy | Keywords: magic wizard adventure
2. Title: Outliers | Genre: Non-fiction | Keywords: motivational improvement
3. Title: Angels and Demons | Genre: Mystery | Keywords: conspiracy illuminati
4. Title: The Phoenix Project | Genre: Non-fiction | Keywords: devops IT improvement

Enter a book you have completed and a rating out of 5

User: 4, 5

Book Advisor:
You gave The Phoenix Project a score of 5/5.

Here an algorithm will be created to give weight to genre and keywords that match and will determine the next most likely book for the user to like based on their current rating and previous ratings. In this example with a 5/5 score based on the genres and keywords provided the next best recommendation would come up as Outliers

Your next book recommendation based on what you like is Outliers.

1.3 REQUIREMENTS IMPLEMENTED IN THIS RELEASE

Supply functional requirements statement that you accomplished for this assignment, together with input where applicable, and output. Please try to state requirement in declarative form, as illustrated in the examples, because here we want to know the functionality intended (*what*, not *how*). The example material should be deleted. Keep in mind that the implementation of these requirements will incorporate *an abstract class*, *inheritance*, *upcasting or downcasting*, and *polymorphism*, and that will probably influence the requirements you choose to implement in this assignment.

1.3.1 Book inputs

Book Advisor should accept a book title, genre, and keywords from a user.

Book Advisor should save data about entered books to an output to persist data.

1.3.2 Book completion

Book Advisor should allow a user to mark a book as completed and give it a rating.

1.4 ILLUSTRATIVE OUTPUT

Provide illustrative output from your application showing that the requirements have been met. Explain what class.method(s) produce it.

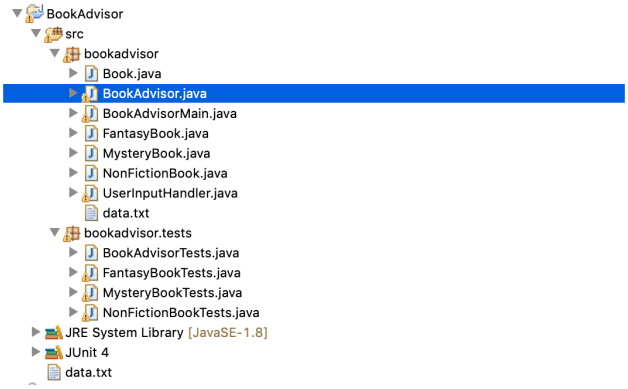
Your response should replace this, as well as the example below. This will be produced by the BookAdvisorMain.completeBook() method. When a user completes a book it will run a recommendation engine to determine which book the user should read next based on Genre and Keywords.

```
=========BookAdvisorDisplayExample==========
Your next recommended book is The Pheonix Project.
It has a 86% match to other books you have read and rated.
```

1.5 YOUR DIRECTORY

Show a screenshot of your directory. This should include a parallel directory of JUnit tests where possible—package-by-package, class-by-class, and method-by-method, except for trivial ones.

Your response should replace this, as well as the example below.
Currently the JUnit tests are in the bookadvisor.tests package. There is an issue with the user input capturing, but the unit tests show that the methods work properly.



1.6 TECHNIQUES IMPLEMENTED

Your implementation should include *inheritance*, *polymorphism*, and *either an abstract class or interface* at least once, and in a manner that is useful to your application. Explain where and how you applied these, using the headings below.

1.6.1 Class model and Sequence Diagram

Identify where you included *inheritance*, *polymorphism*, and *abstract classes* or *interfaces* in your class model. Make classes and members *static* or not as per their intended usage. To do this use tools (e.g., Visio), PowerPoint, or a combine models as in this example (which you are free to cut and paste from). Insert indications in red (as in this example) to show where the three features below apply.

The class model for Book Advisor is located in the Book Advisor RUML.xlsx file included.

1.6.2 Code showing an abstract class or interface.

Show the relevant code (only) and explain why an abstract class or interface is appropriate here. It should be clear where the code is located (class and method).

Your response should replace this, as well as the example below.

For now there are three types of books, FantasyBook, MysteryBook and NonFictionBook.

There are only two kinds of fragments, so there is no need for the *Fragment* base class to be concrete since there will be no generic Fragments.

```
package bookadvisor;


import java.util.List;
import java.util.Arrays;


// An object that represents a book


public abstract class Book {
    private String title;
    private String genre;
    private String[] keywords;
    private int rating;


    public Book(String title, String keywords){
        this.title = title;
        this.keywords = keywords.split(" ");
```

```
    }

    public String getTitle(){
        return this.title;
    }

    public abstract String getGenre();

    public String[] getKeywords(){
        return this.keywords;
    }

    public int getRating(){
        return this.rating;
    }

    public void setRating(int rating){
        this.rating = rating;
    }
    public abstract double getGenreSimilarityWeight(Book book);
}
```

1.6.3 Code showing polymorphism.

Show the relevant code (only) and explain why *polymorphism* is appropriate here. Recall that polymorphism is implemented in one of two ways – overriding methods in subclasses or overloading methods in the same class where the method signatures are different – and allowing the language runtime to dynamically invoke the correct method. It should be clear where the code is located (class and method).

The method *instantiateBooks()* in the BookAdvisor class uses polymorphism to use a switch statement to derive the genre of a book to create the proper type. This method is a common

method that gets used in two places. One place is when the user adds a new book to the list to instantiate the right type. It also gets used when loading books from the save file to add them to the ArrayList data structure.

```
private Book instantiateBook(String title, String keywords, String genre){
    // intent: use polymorphism to create the correct instance of a book object
    // postcondition: book is created using the correct type
    Book bookToAdd;
    switch(genre){
        case FantasyBook.genreName:
            bookToAdd = new FantasyBook(title, keywords);
            break;
        case MysteryBook.genreName:
            bookToAdd = new MysteryBook(title, keywords);
            break;
        case NonFictionBook.genreName:
            bookToAdd = new NonFictionBook(title, keywords);
            break;
        default:
            bookToAdd = null;
            break;
    }
    return bookToAdd;
}
```

1.6.4 Code showing upcasting or downcasting.

Show the relevant code (only) and explain why upcasting or downcasting is appropriate here. It should be clear where the code is located (class and method).

To be able to get a list of books of a specific genre this method *getBookOfGenre()* loops through the books and returns a list of the genre passed as a parameter which get downcast to the specific type of book which is passed as a parameter.

```
public <T extends Book> ArrayList<T> getBookOfGenre(String genreName){
    ArrayList<T> booksInGenre = new ArrayList<T>();
    // intent: downcast books to books of a specific genre
    // postcondition: all of the books in the genre that have been added get returned here
    for(Book book : this.books){
        if(genreName == book.getGenre()){
            booksInGenre.add((T) book);
        }
    }

    return booksInGenre;
}
```

1.7 YOUR CODE

Unless your facilitator requests another method, copy your Eclipse project to your file system, zip it, and attach it. Please contact your facilitator in advance if you want to request an alternative means.

<Your response here>

1.8 Instructor's Evaluation

Criterion	D	C	B	A	Letter Grade	%
Technical Correctness	No justification of correctness	Technically mostly correct; tested	Implementation correctness well justified; well specified and tested	Implementation correctness thoroughly justified throughout by precise block- and line-specifications and tests.		0.0
Clarity in Presentation	Unclear	Somewhat clear; some comments	Clear with a few exceptions. Well commented.	Entirely clear throughout; very well commented at high and low levels.		0.0
Depth and Thoroughness of Coverage	Shallow or superficial coverage of most topics	Satisfactory depth and thoroughness	Evidence of depth and thoroughness in covering most topics	Evidence of depth and thoroughness in coverring all topics		0.0
				Assignment Grade:		0.0
The resulting grade is the average of these, using A+=97, A=95, A-=90, B+=87, B=85, B-=80 etc. The maximum is 100.						
To obtain an A grade for the course, your weighted average should be >93. A-:>=90. B+:>=87. B:>83. B-:>=80 etc.						

Appendix 1 (if needed; should be referenced above, and will be read as-needed only)

Appendix 2 (if needed; should be referenced above, and will be read as-needed only)