Lab 9 – Creating and Using Maps

In this lab, you will implement a ***BookCollection*** application using Java's ***HashMap*** class. You'll then create and test your own tree-based map and use it in your ***BookCollection*** application.

Unzip the lab directory into your ***CS200/labs*** and open it to find a data file called ***sampleLibrary.BookCollection***, along with five java files: ***BookCollection***, ***BookDescription***, ***InnerIteratorForTreeMap***, ***TreeMapTest***, and ***TestClassStats***. You'll use the code in class ***InnerIteratorForTreeMap*** as an inner class in your map class. For now, keep all files in your lab folder.

We'll continue with the expectation that you'll complete the lab WITH MINIMAL to NO assistance from your lab instructor or TA

Problem 1 (of 3): Building the BookCollection application

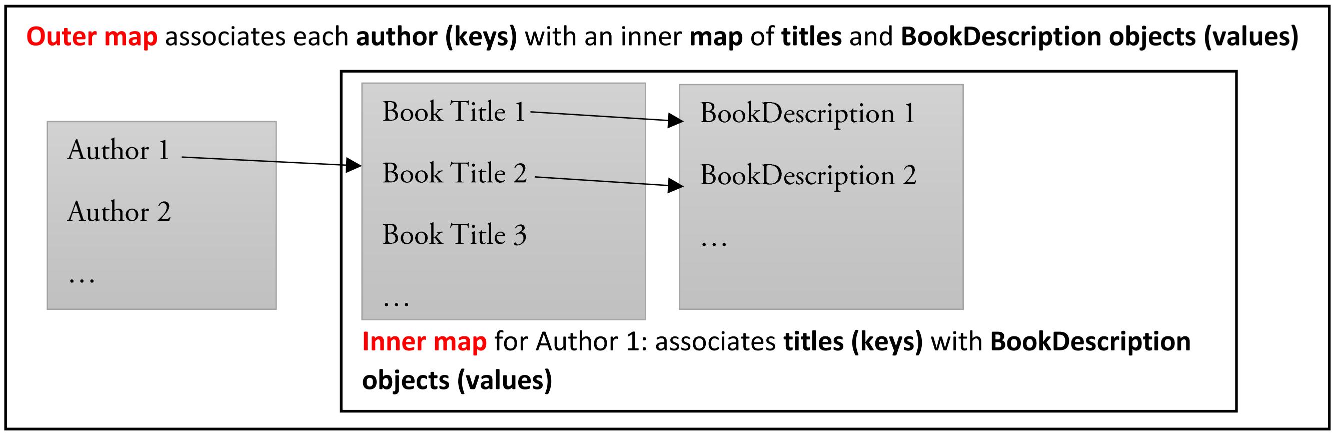
In this lab problem, and unlike most previous ones so far, you have *some* flexibility to choose how to implement certain functionalities described here. You'll be graded on your demo and the clarity of your driver program output described in ***Part 1.3***. Please read all the problem before starting.

The core of the book collection is a ***HashMap<****String , **HashMap**<String , BookDescription>***>**. Notice that we're dealing with nested maps here (a map used as the value of another map). And you thought you've seen it all!!

An outer map that associates ***String*** author names as keys with inner maps as values.

An inner map that associates ***String*** book titles as keys with ***BookDescription*** objects as values.

Below is a visual illustration.



Outer map associates each **author (keys)** with an inner **map** of **titles** and **BookDescription objects (values)**
Author 1
Author 2
…
Book Title 1
Book Title 2
Book Title 3
…
BookDescription 1
BookDescription 2
…
Inner map for Author 1: associates **titles (keys)** with **BookDescription objects (values)**

First, you'll develop the ***BookDescription*** and ***BookCollection*** classes; the next step would be to develop an application that manages the collection. For your maps, you'll first use Java's ***Map*** interface and ***HashMap*** class, which are documented on the official Java API pages:
http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html

Part 1.1: The BookDescription class

Complete the ***BookDescription*** class which should contain the following private instance fields:

author –the author

title –the title of the book

publisher–the name of the publisher

date–the year of publication

rating–your own single-character rating system for the books

pages–number of pages

description–a description of the book

Create multiple constructors for your class: default constructor, constructor that takes all parameters, constructor that takes only some parameters including author and title, etc. In addition, create an accessor/getter method for <u>*every*</u> instance field, a ***toString*** method, a ***setDescription*** method and a ***setRating*** method.

The ***toString*** method should format the string to include newline characters so that it matches the text format in the ***sampleLibrary.BookCollection*** file (please look closely at this file which contains book descriptions in the appropriate format).

You may choose to have more than the specified mutator/setter methods, but only for fields that might change over time; the author and title of a book presumably never change, so those fields should not have mutator methods.

Part 1.2: The BookCollection class

Once you have the ***BookDescription*** class complete, move on to the ***BookCollection*** class which contains the following instance variable:

```
private HashMap<String, HashMap<String, BookDescription>> library
```

A ***BookCollection*** contains a map of ***BookDescription*** objects that are accessed by *author* and *title*. That is, the ***library*** instance variable maps <u>authors</u> to their <u>books map</u>, each of which maps <u>titles</u> to their <u>***BookDescription*** objects</u>. To get the ***BookDescription*** object of a particular book, you first must get the books written by the author and then get the title from among those books.

Your ***BookCollection*** class should have methods and constructors that enable you to accomplish each of the following:

(1) ***BookCollection()***: a default constructor that creates a new empty collection

(2) ***BookCollection(String** fileName**)***: a second constructor with a ***String*** filename parameter that creates a new collection by reading books from a specified file formatted as shown in file ***sampleLibrary.BookCollection***

(3) ***allAuthors ()***: returns an ***Iterator<String>*** that yields author names (which are the ***keys*** in the outer map), one-by-one (note that the outer map already has an iterator method for us to call)

(4) ***allAuthorTitles (String** author**)***: returns an ***Iterator<String>*** that yields title names (which are the ***keys*** in the inner map), one-by-one, for the given author (again, note that the inner map already has such a method; if there is no entry for the specified author, the method should return ***null***)

(5) ***get(String** author**, String** title**)***: gets a particular ***BookDescription*** object for a given author and title

(6) ***add(BookDescription** bd**)***: adds the given ***BookDescription*** object to the collection (i.e., to the author's inner Map)

(7) ***remove(String** author**, String** title**)***: removes a ***BookDescription*** object given its author and title

(8) ***update(String** author**, String** title**, String** newDescription**)***: updates the description of a ***BookDescription*** object given its author and title

(9) ***write(String** fileName**)***: writes the collection to a text file formatted as shown in file ***sampleLibrary.BookCollection*** to allow us to continue the application from a saved point

(10) ***isEmpty()***: checks if a book collection is empty

(11) ***size()***: gets the size of a book collection (number of books for all authors)

File ***sampleLibrary.BookCollection*** shows a sample of textbook descriptions in the format appropriate for this lab. Your second constructor should be able to read this file and should read any file with that format. The ***write*** method should produce files in the same format. The ***toString*** method of the ***BookDescription*** class should facilitate this process.

Part 1.3: The driver program

Create a driver program that uses methods from class ***BookCollection*** to demo each of the functionalities below. You may simply include a ***main*** method in your ***BookCollection*** class for this purpose or create a new driver program similar to and using ones from earlier labs as a model; note, however, your driver program doesn't have to be interactive.

You are expected to be very thorough in your demo. Specifically, I want to see adequate output from your driver program to easily demonstrate the following functionalities:

(1) create a collection by reading books from any file similar to the provided file ***sampleLibrary.BookCollection***

(2) use the collection to iterate over and display all author names–ONLY author names

(3) use the collection to iterate over and display book titles for a specific author–ONLY titles

(4) update the description of one of the books for any of the authors, and then, display all books for that author to show the updated value

(5) add a new book to your collection and then, display books for all authors to show the new book

(6) remove a book from your collection and display remaining books for all authors

(7) write the collection to a new file and check that file has been written properly (you should be able to read this file in (1))

Problem 2 (of 3): Implementing and testing a tree-based map class

Recall that a map is a set of ***<key, value>*** pairs; every key in the map is unique, but the same value may be associated with more than one key. In this problem you will implement a tree-based map class using your ***<FOO>LinkedBinarySearchTree*** class from **Lab 08**. In other words, you'll make a binary search tree object store ***<key, value>*** pair elements and behave like a map (but not a hash map).

Part 2.1: Creating the tree map class

Keep files ***TreeMapTest*** and ***TestClassStats*** in your lab folder. Open ***TreeMapTest*** and change all instances of ***<FOO>*** and ***<foo>*** to the appropriate uppercase or lowercase initials you are using. Next, rename file ***InnerIteratorForTreeMap*** to ***<FOO>TreeMap.java*** using your initials in place of ***<FOO>*** and move the file to your ***<foo>structures*** directory.

Open this file and create an outer class with the following header (place the outer class above inner class ***MapIterator***). Study interface ***ZHMap*** on *https://faculty.csbsju.edu/irahal/SW/ZHStructures/*

```
public class <FOO>TreeMap<KeyType extends Comparable<KeyType>, ValueType>
                          implements ZHMap<KeyType, ValueType>
```

Your tree map class will have only one instance variable as shown next–as always, don't change names of fields or methods; otherwise, you're likely to encounter failed JUnit tests.

```
private ZHBinarySearchTree<ZHComparableKeyPair<KeyType, ValueType>> innerTree
```

Its only constructor should create an empty map, so it doesn't need any parameters, but it should initialize the ***innerTree*** to be an empty ***<FOO>LinkedBinarySearchTree*** (use your initials instead of ***<FOO>***).

The documentation for the ***ZHMap*** interface shows all the methods you need to implement in your class.

Part 2.2: Completing the tree map class

Now, you will complete the code for the tree map class so that it correctly implements the specified methods. Several of the methods need to deal explicitly with the ***ZHComparableKeyPair*** class so please study its documentation here: *https://faculty.csbsju.edu/irahal/SW/ZHStructures/*. In a nutshell, this class allows us to treat a (***key,value***) pair as a single object which we'll store as an element in the ***innerTree*** instance variable.

Here are brief descriptions of how these methods should work:

`isEmpty` and **`size`**: These should call corresponding methods using the ***innerTree*** instance variable.

`iterator`: This method should return a new instance of the inner ***MapIterator*** class supplied with this lab. You can see that the code for the iterator is straightforward; it just wraps a ***KeyType*** iterator around a ***ZHComparableKeyPair*** iterator associated with the ***innerTree*** instance variable allowing us to iterate over the keys in the map.

`contains`, `get`, and `remove`: These methods should construct a ***ZHComparableKeyPair*** object with the desired ***key*** and a *null* ***value***, and call corresponding methods on the ***innerTree*** instance variable with the created ***ZHComparableKeyPair*** object as parameter; methods here will return the values returned by calls on the ***innerTree*** instance variable.

`put`: This method first constructs a ***ZHComparableKeyPair*** with the specified ***key*** and ***value***, and then calls the ***innerTree***'s ***get*** method; depending on the value returned by the ***get*** method, the method should either (1) call the tree's ***add*** method and return ***null*** (for a new pair) OR (2) call the returned pair's ***setValue*** method with the specified new value and return the old value.

Part 2.3: Testing the tree map class

Once you have a complete tree map class, use the supplied test program to test your code. Show your fully documented code and the testing to the lab instructor or TA before you proceed.

Problem 3 (of 3): Using your map in the BookCollection application

Make copies of files ***BookDescription*** and ***BookCollection*** from ***Problem 1*** and rename them ***BookDescriptionV2*** and ***BookCollectionV2*** to distinguish them from the previous version. If you created a separate class for your driver program, then do the same for that file as well. Open the renamed files and make the necessary changes so that your application now uses your map class instead of java's ***HashMap***. In general, you'll need to import the ***zhstructures*** and ***<foo>structures*** packages, change the interface names to ***ZH*** interfaces and change the class names to ***<FOO>*** classes.

Test your changes to make sure that your new ***BookCollection*** application still works. It should be able to read the same files that the original read or wrote and to write files that are readable by the original version.

Have fun.