1 Introduction

Java Collections are very nice data structures. There a lots of built in methods for them and we are going to take a look at a few things before we start playing with Maps. An ArrayList is part of the Java Collections framework. Download the BookStoreV2.zip and unzip it. Open the project in NetBeans.

This is the same application that we wrote the first week of school, except it is written using an ArrayList instead of an array. If you look at the new readInventory() method, right after the file is closed, the ArrayList is sorted.

Now instead of using a linear search we can use a better one!

2 Binary Search

The binary search will only work on sorted lists. A feature of this search is that it is super fast. Another feature is that if it doesn't find what it is looking for it returns the negative integer that represents where it would be if it were in the list. The binary search algorithm begins by setting values for low and high, which would be 0 and .size()-1 to start with. Then it computes the index of the middle element to see if it (1) matches, (2) is less than, or (3) is greater than. If the thing we are looking for is less than the element in the middle, then we reassign high to middle-1. We don't need to look at the previous middle element again. Now we have only 1/2 of the original list to search. Again, we calculate the middle element and look at the relationship to the thing we are looking for. Move your cursor to the end of readInventory() We will put the binarySearch() method between readInventory() and updateInventory(). Here is the code for the binary search method.

```
/**
 * Binary search is used on sorted data only!  It effectively cuts the
 * search area in half at each iteration. The number of comparisons for
 * a search is (log2 n) which is very fast.  The search will only take
 * 13 iterations for a list of size 10,000.
 * @param keyToFind the String we are looking for
 * @return the index where the item was found or the place where it
 * would be if it were in the list.
 */
public int binarySearch(String keyToFind){
   int low = 0;           // starting index
   int high = inventory.size()-1; // ending indes
   int mid = 0;           // index of the middle element
   int iterations = 0;    // counter for the number of searches
   boolean found = false; // we haven't started looking yet
```

```
    while (low <= high & !found){
        mid = low + (high - low ) / 2 ;
        if (inventory.get(mid).getIsbn().compareTo(keyToFind) == 0){
           found = true;
        } else if (inventory.get(mid).getIsbn().compareTo(keyToFind) > 0){
           high = mid - 1;
        } else{
           low = mid + 1;
        }

        iterations++;
    }
    System.out.println("number of iterations: " + iterations);
    if (!found){
         mid = - mid-1; // where it would be if it was here
    }
    return mid;
}
```

Test it and make sure that it still runs successfully. We should have 178 books.

3 Now for using a Map

Save the project. Close the project in NetBeans. In Windows Explorer (or Finder) make a copy of the whole project folder that we can change to use a Map instead. Rename the folder by right-clicking on it and name it BookStoreV3. Now open that project in NetBeans.

Right-click on the project name in the pane on the left and rename the project BookStoreV3. Now right-click the package and refactor =>rename it bookstorev3 (all lower case). Finally right- click the BookStoreV2.java file and refactor =>rename it BookStoreV3. A map is a data structure that uses a key, value pair for each entry in the Map. The key must be unique. The value can be any object or even another data structure.

Open the BookStoreV3.java file in the editor and change the property from an ArrayList to a Map.

```
public class BookStoreV3 {

    private Map <String, Book> inventory = new TreeMap<>();
```

A TreeMap is in Key order. Since our keys are the isbn numbers, that is the order that it will maintained as we add to the map.

In readInventory() make the following change to the line inventory.add()

```
inventory.put(isbn, new Book(isbn, title,
```

```
                                author, pages, price,
                                numberOnShelf));
```

and delete the call to Collections.sort().

Because of the unique way that it organizes the keys, we don't need to search this data structure at all. You can delete the binarySearch() method all together.

In updateInventory() change the if statement in the while loop to put the new book in the map:

```
if (isbn.length() == 13 && numberInShipment >= 0) {
   if (inventory.containsKey(isbn)) {
         Book b = inventory.get(isbn);
         b.addToNumberOnShelf(numberInShipment);
         inventory.put(isbn, b);
         booksUpdated++;
   } else {
         inventory.put(isbn, new Book(isbn, title,
                  author, pages, price,
                  numberInShipment));
         newBooks++;
   }
}
```

NOTE: you MUST check to see if what you are looking for is in the list BEFORE you attempt to "get" it. That is: you must use if (inventory.containsKey(isbn)) before using inventory.get(isbn). You can't get it if it isn't there. You would get a return of null and you can't call the addToNumberOnShelf() method if there is no object.

Delete the call to Collections.sort().

In the writeInventory() method make the following change to the for loop:

```
         for (String isbn : inventory.keySet()){
             out.println(inventory.get(isbn));
         }
```

The keySet() is the set of keys from the map in isbn number since our map is a TreeMap.

4 Time to Run and Test the Whole Application

Time to click the green arrow on the tool bar and see if see if we have 178 books in the updated inventory. Make sure you understand everything that we did.

5 Evil Hangman - the program cheats!

Evil Hangman is a game where we will program the computer to cheat. It will be almost impossible to win the game. This version will select a random word only to get a length, then make a set of all the words in the dictionary that are that length.

Each time a player chooses a letter all words in the set with that letter will be removed from the set.

If there comes a time that the set only holds one word it will be 'chosen' and play will continue normally. Odds are that won't happen before the 7th wrong answer.

Download EvilHangman.zip and unzip it. As it is right now, it is the classic game of hangman. Take some time to look at the code that is there. Read the comments and the code.

Play a game. Debugging is 'true' so there will be lines that print to help you test and play that would not print if we change the property `debugging` to false. We will be debugging for a while, so leave the property as it is.

Add a new property to the list of properties; the set of possible words.

```
private TreeSet<String> possibilities;  // NEW
```

Move down to the setRandomWord() method. We no longer need to remove the word from the dictionary because it really won't be the chosen word. Change it as follows:

```
    public void setRandomGameWord() {
        Random random = new Random();
        int index = random.nextInt(dictionary.size());
        gameWord = dictionary.get(index);
    }
```

Use Source — Format to fix the indentation.

Below that method let's add a new one that creates the set of possible words by looking at the length of each word in the dictionary. Calling a method "costs" more in computing power than accessing a primitive variable.

```
/**
 * Creates a set of words with the same length as the random word
 */
    public void createSet() {
        possibilities = new TreeSet<String>();
        // local int so we don't call the length method for
        // the gameWord inside the loop
        int len = gameWord.length();
        for (String word : dictionary) {
            if (word.length() == len) {
                possibilities.add(word);
            }
        }
        if (debugging){
            System.out.println("There are " + possibilities.size()
                    + " words in the set");
        }
    }
```

Use Source — Format to fix the indentation.

We need to add a call to createSet() in initializeForGame(), right after the call to `setRandomGameWord()`. Edit it to match this:

```
public void initializeForNewGame() {
    setRandomGameWord();
    createSet();    //  NEW
    wrongGuesses = 0;
```

Use Source — Format to fix the indentation.

The method scoreLetter() is the one that changes the most. It will now remove all words in the set of possibilities that contain the letter that the user guesses. Here is the complete method with the changes.

```
/**
 * Removes all words containing the guessed letter from the set of
 * possible words and increases wrongGuesses by one.  When the
 * possible words gets down to 1 the game must use that last word
 * as the gameWord and normal scoring of the guesses would proceed
 * from that point. Normal Scoring: Finds the current guess in the
 * gameWordArray and replaces the corresponding '*' with the letter
 * in displayWord. If the letter is not found it increments wrong
 * guess counter.
 *
 * @param letter the letter the player guessed
 */
public void scoreLetter(char letter) {
    boolean letterFound = false;
    // necessary for the contains() method used below
    String charSequence = Character.toString(letter);
    guessedLetters.add(letter); //  duplicates are ignored
    // everything from here to the for loop is new
    // if the set contains more than one word...
    if (possibilities.size() > 1) {
        // create an iterator for the set there are no indexes in a
        // set this is how to cycle through all the members of the set
        Iterator<String> iterator = possibilities.iterator();
        // delete all words that contain the letter that was guessed
        while (iterator.hasNext() && possibilities.size() > 1) {
            if (iterator.next().contains(charSequence)) {
                iterator.remove();
            }
        }
    }
    if (debugging){
        System.out.println("words left in set: "
```

```
                            + possibilities.size());
        }
        // if we empty out the set in the loop above, we have to take
        // the word that is left as the gameword.
        if (possibilities.size() == 1){
            gameWord = possibilities.pollFirst(); // removes from the set
            gameWordArray = new char[gameWord.length()];
            gameWordArray = gameWord.toCharArray();
            if (debugging){
                 System.out.println("Forced to choose a word. It is: "
                    + gameWord);
            }
        }
        // if the set is empty, score the word the same way we did
        // in the original hangman - the for loop is old code.
        if (possibilities.size() == 0){
            for (int i = 0; i < gameWordArray.length; i++) {
                if (gameWordArray[i] == letter) {
                    displayWord[i] = letter;
                    letterFound = true;
                }
            }
        }
        if (!letterFound) {
            wrongGuesses++;
        }
    }
```

Click next to the red dot by the Iterator line and include java.util.Iterator

Use Source — Format to fix the indentation.

We need to add one line to the run method so that it has a word that has none of the guessed letters in it. Scroll down somewhere around line 235 there is an if statement:

```
if (wrongGuesses == MAX_WRONG_GUESSES) {
    gameWord = possibilities.pollFirst(); // NEW
    dictionary.remove(gameWord);  // NEW
    printTheGallows();
    System.out.println("Sorry, you lose! The word was " + gameWord);
}
```

Add the two lines marked "NEW" to the if statement in your program.

No other changes are necessary. Play a game! Don't let your children play this, it really isn't fair.

6 What did you learn?

If you downloaded a Lab Project for today's lab, you will find a LabWriteUp.txt document inside it. If the lab instructions asked you to create a project for the lab, download the writeup document from Brightspace and save it to your project folder for today.

Open it in a text editor like notepad (do not use Word!) and answer the questions so that what you learned from the lab is clear. You MUST understand what you have done and why. If you don't understand, ask! Close all open projects in NetBeans.

In Explorer or Finder Right-click on the project folder and choose Send To Compressed file. Name it with your last name and ONLY your last name. Upload that to Brightspace. There are Videos on how to compress folders on both Windows and Mac in the How Tos in the Welcome Content area. Do NOT use the zip utility in NetBeans. The Archive MUST be a .zip file.

7 How Labs are Graded

Labs are graded as Pass/No-Pass and can be resubmitted one time.

Pass	Projects are done according to the instructions and the Writeup shows a complete understanding of today's lab concepts
No-Pass	Projects are not done according to the instructions and/or Writeup shows very little understanding of the lab concepts