CS 250 - Fall 2021 - Lab 9: Collections Framework

Available: Oct 27, 2021
Due date: **Nov 1, 2021, 11:59 PM**

In this lab, we will continue to practice different abstract data types which belong to the java.util.Collections package in Java.

Objectives:

The objectives of this lab are:

- To understand different abstract data types in the Collections package: TreeMap, Stack, Queue, etc.
- To practice coding abstract data types.

Prerequisite and Resource:

- Read Java API documentation about Stack, Queue, PriorityQueue, etc.

Part 1: Count the Occurrences of Words

In the first part of this lab, we will develop a Java application that uses Java's TreeMap class to store the occurrences of words in an input string. The TreeMap class is very similar to the HashMap class in that they both implement the Map and Collection interfaces. Similar to the HashMap, you can insert entries that are pairs of a key and value. The map cannot contain duplicate keys, and each key is mapped to one value. In this application, the user is asked to input a sentence. Then the program will count and print the number of occurrences of words in the input string.

1. Create a new project named Lab9 and a class named CountOccurenceOfWords.

2. Copy the following skeleton to replace the generated code in Eclipse:

```
public class CountOccurenceOfWords {
	public static Map<String, Integer> countOccurences(String sentence) {
		//TODO add code below
	}

	public static void main(String[] args) {
		//TODO add code below
	}
```

```
}
```

3. **Complete the countOccurences(String sentence) method**. This method receives a String input and needs to return a Map of String and Integer, which stores the number of occurrences of each word in the sentence. Complete this method with the following hints:

- Create a Map<String, Integer> object as an instance of the TreeMap class
- Split the sentence into a String array of words
- For each String element (word) in the String array:
    - String str = trim leading and ending space from the word
    - Convert str to lowercase
    - If str is not an empty string
        - If str is not an existing key in the map, pun an entry (str, 1) to the map. This operation inserts an entry word with a counting of 1 occurrence into the map.
        - Else, get the value corresponding with the key=str in the map. Increase the value with 1 (increase 1 to the count of str). Finally, put back the entry (str, value) into the map.
- Return the map object

4. **Complete the main() method** with the following instructions:

- Create a Scanner object
- Prompt a text "Enter a sentence: "
- Use the Scanner object to read the sentence into a String variable named line
- Get the Map object (which contain the occurrences of words in the input sentence) by calling the method countOccurences()
- Print the above Map object with each entry on each line

Test the program with some of the famous quotes below:

"A life spent making mistakes is not only more honorable but more useful than a life spent doing nothing." - George Bernhard Shaw

"Everybody is a genius. But if you judge a fish by its ability to climb a tree, it will live its whole life believing that it is stupid." — Albert Einstein

"We think sometimes that poverty is only being hungry, naked and homeless. The poverty of being unwanted, unloved and uncared for is the greatest poverty." — Mother Teresa

Part 2: Palindrome Checker

A **palindrome** is a word or phrase that reads the same forward and backward, ignoring blanks and considering uppercase and lowercase versions of the same letter to be equal. For example, the following are palindromes:

• dad

• Anna

• radar

• tacocat

• Able was I ere I saw Elba

In the second part of this lab, we develop a Java program that accepts string input from the user and check if the input string is a palindrome or not. We will use Stack in Java Collections framework to solve this problem.

1. Create a new class named PalindromeChecker and copy the following skeleton code:

```
public class PalindromeChecker {
	/**
	 * This method returns true if the given string str is a palindrome
	 */
	public static boolean isPalindrome(String str) {
		//TODO: Add code below
	}

	/**
	 * main() application method
	 */
	public static void main(String args[]) {
		//TODO: Add code below
	}
}
```

2. **Complete the isPalindrome() method**. This method receives an input String and returns true if it is a palindrome. We use Stack to implement this method. The idea here is we push the characters in the first half of the input string to a stack, then sequentially pop each character and compare with each character in the second half of the input string. If there are any unmatch, return 0. We skip the check on the middle character if the length is odd. Check the following example:

- “dad”: push ‘d’ to the stack -> skip ‘a’ -> pop ‘d’ and check with the last character => match

- “anna”: push ‘a’ then ‘n’ -> then pop ‘n’ and match with ‘n’, then pop ‘a’ and match with ‘a’ at the last character

Use the following hints:

- stack = Create a Stack of Character
- String tmp = Convert str to lower case
- Calculate the half-length of the input string: mid = tmp.length / 2
- Use a loop to push the first character to the middle character of tmp to stack (skip the middle character if the length of tmp is odd)
- i = mid + 1 if tmp.length is odd else mid
- While i < tmp.length:
    - ch = Pop the character from the stack
    - If ch is not equal to the character at the index i of tmp, return false
    - Increase 1 to i
- Return true

3. Add code to the **main()** method to test the **isPalindrome** method with the following code:

- Create a Scanner object
- Prompt a text “Enter the string: “
- Use the Scanner object to read an input line as String
- Calling the isPalindrome() method on that input string
- Print the check result

Part 3: Student Course Organizer

In this part of the lab, we develop a Java program to organize the list of course registration of the students in the format “StudentID CourseNumber”, each registration is on a separate line. The user will input this information without order. For example, if student 101 is in CS100 and CS200 while student 102 is in CS105 and MATH210 then the list might look like this:

101 CS100

102 MATH210

102 CS105

101 CS200

The user terminates the input with -1.

To solve this problem, we can use the HashMap class to map from an Integer (the student ID) to an ArrayList of type String that holds each class the student is enrolled in.

1. Create a new class named CourseRegistration and copy the following skeleton code:

```
public class CourseRegistration {
        /**
         * main() application method
         */
        public static void main(String args[]) {
                //Add code below
        } // main()
}
```

2. Add code to the **main()** method to complete this application:

- Create a Scanner object
- Create a HashMap object that maps Integer (student id) to an ArrayList of String (list of course names):

  HashMap<Integer, ArrayList<String>> students = new HashMap<Integer, ArrayList<String>>();

- Prompt a text "Enter the list of course registration: "
- while(true) {
  - line = Use the Scanner object to read an input line as String
  - Split line into a string array by space
  - student_id = Parse the first element in the string array into Integer
  - If (student_id == -1), exit the loop (break)
  - course_name = the second element in the string array
  - If student_id is not a key in the HashMap students
    - ArrayList<String> arl = Create an ArrayList of String and put course_name into it
    - Put a new entry (student_id, arl) into the HashMap students
  - Else,
    - ArrayList<String> arl = Get the value corresponding to student_id in the HashMap students
    - Append course_name into arl if such a course name does not exist in the list yet. Otherwise, output an error.

        - Put back the entry (student_id, arl) into the HashMap students.
- }
- For each student_id (key) in the HashMap students
    - ArrayList<String> arl = Get the value corresponding to student_id in the HashMap students
    - Print the student_id and the list of courses in the array list arl

Lab Assignment

Complete the following assignment of this lab:

1. Design and write a program that uses a Stack to determine whether a parenthesized expression is well-formed. Such an expression is well-formed only if there is a closing parenthesis for each opening parenthesis. For example, the following examples are the correct ones: “()”, “a(c)b”, “(a+b)”, “(a+b)(c)”, “(())”

Hint: push to Stack if getting “(“, pop and check for a match if getting “)”

2. Extend the application in the third part to allow the user to enter a course name and print the list of student IDs who registered for that course. You may need to implement a while loop so the user can enter the search course name multiple times until a specific text is entered, i.e., “-1” (then exit the loop).

Hint: For each student id in the HashMap, add the student id into the result list if the target course exists in such the student’s course list.

3. Extend the application in the third part to print the course that has the maximum number of enrolled students.

Lab 9 Result Submission:

You need to submit the results of the following sections to the D2L assignment item of Lab9:

- Complete the required Lab Assignment

- Compress the whole Eclipse source code of your Lab9 project in zip format using the Export function