We confirm that we did not receive any sort of help on any of these questions worth noting -> the most help we received this week was Professor Jimenez telling use to clear the stack after using it in testing.

- Howie Youngdahl, Havin Lim

1. The keys and values are most likely associated with the HashMap class.
2. Hashmap is faster, but TreeMap is ordered. In this case, we would want to use the hashmap class because ordering is unnecessary, and the amount of datapoints would direct us to using hashmap because its faster.

```
// Anagram.java


package lab17;


import java.util.TreeSet;

import java.util.Arrays;


public class Anagram{


public static String sorty(String str) {

str = str.toLowerCase();

char[] chars = str.toCharArray();

Arrays.sort(chars);

String ans = "";


for (char c : chars) {

ans += c;

}

return ans;

}


public static Boolean areAnagram(String str1, String str2) {

return sorty(str1).equals(sorty(str2));
}




public static void main(String[] args){
System.out.println(sorty("havin"));
System.out.println(sorty("vianh"));
System.out.println(areAnagram("havin", "vianh"));
}
}


// AnagramFinder.java

package lab17;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class AnagramFinder {
```

```
public static void main(String[] args) {
File file = new File(args[0]);
Scanner sc;
try {
sc = new Scanner(file);
while(sc.hasNextLine()) {
String line = sc.nextLine();
addToMap(line);
}
} catch (FileNotFoundException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
for ( String key : hashy.keySet() ) {
  if(hashy.get(key).size() > 0) {
        ArrayList<String> current = new ArrayList<String>(hashy.get(key));
        if (current.size() > 1) {
              System.out.println(current);
        }
  }
}


}
static Map<String, ArrayList<String>> hashy = new HashMap<String, ArrayList<String>>();
public static void addToMap(String str){
if(hashy.containsKey(Anagrams.sorty(str))){
ArrayList<String> current = hashy.get(Anagrams.sorty(str));
current.add(str);
}
else {
ArrayList<String> newCurrent = new ArrayList<String>();
newCurrent.add(str);
hashy.put(Anagrams.sorty(str), newCurrent);
}

}
}
```

// **SECOND LAB OF THE WEEK:**

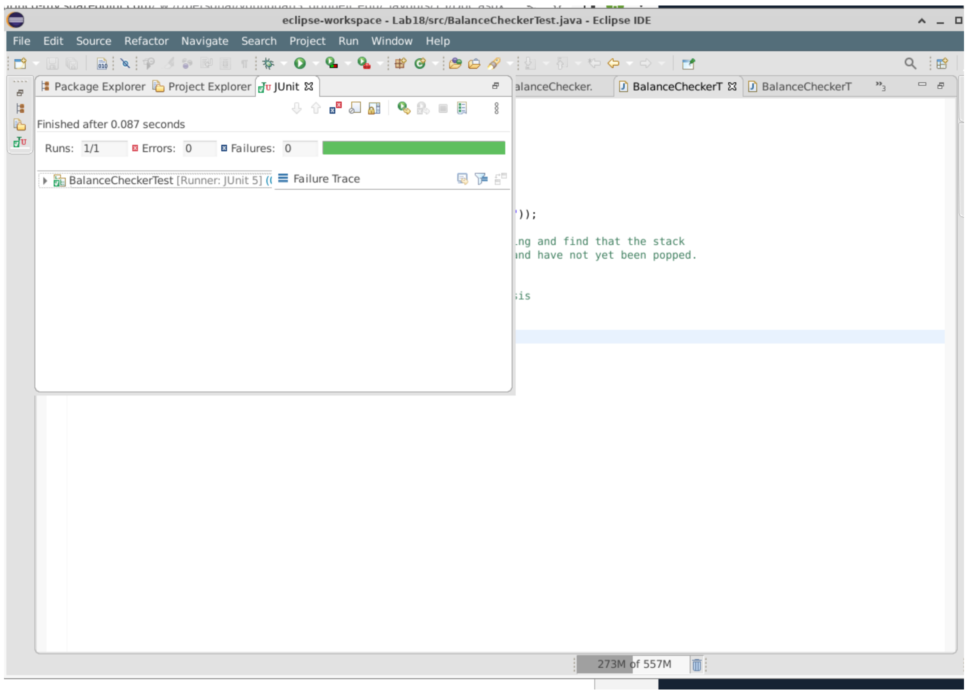
// BalanceChecker.java

```
import java.util.Stack;

/**
*
* @author Howie Youngdahl, Havin Lim
*
*
*/
public class BalanceChecker {
public static Stack<Character> stacky = new Stack<>();

/**
```

```
 *
 * @param a
 * @return
 */
public static boolean checkBalance(String str){
stacky.clear();
char[] arraie = str.toCharArray();
for(char c : arraie) {
if(c == '{' || c == '(' || c == '[') {
stacky.push(c);
}
else if (c == '}'){
if(stacky.isEmpty() || stacky.pop() != '{') {
return false;
}
}
else if (c == ')'){
if(stacky.isEmpty() || stacky.pop() != '(') {
return false;
}
}
else if (c == ']'){
if(stacky.isEmpty() || stacky.pop() != '[') {
return false;
}
}
}
return stacky.isEmpty();
```

```
}

}

// BalanceCheckerTest.java

import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;

class BalanceCheckerTest {

@Test

void test() {

// When the string is balanced

assertEquals(true, BalanceChecker.checkBalance("(a[{bc}(de)])"));

// When the string is just wrong and reach the end of the string and find that the stack

// still has some characters in it that were pushed early on and have not yet been popped.

assertEquals(false, BalanceChecker.checkBalance("({{{{{}}}"));

// When it meets a right bracket but pops out a left parenthesis

assertEquals(false, BalanceChecker.checkBalance("(}"));

// When it meets a right bracket but the stack is empty

assertEquals(false, BalanceChecker.checkBalance("}"));

}

}
```

```
// BalanceCheckerTester.java

public class BalanceCheckerTester {
public static void main (String[] args) {
System.out.println("Here are our findings, kind grader:");

String str1 = "(a[b)c]";
String str2 = "(a[b(c)]";
String str3 = "[a(b(c))d]]";
String str4 = "(a[{bc}(de)])";
String str5 = "(a{bc}d)";

System.out.println("For the first string, is it balanced?  " + BalanceChecker.checkBalance(str1));
System.out.println("For the second string, is it balanced? " + BalanceChecker.checkBalance(str2));
System.out.println("For the third string, is it balanced?  " + BalanceChecker.checkBalance(str3));
System.out.println("For the fourth string, is it balanced? " + BalanceChecker.checkBalance(str4));
System.out.println("For the fifth string, is it balanced?  " + BalanceChecker.checkBalance(str5));

}
}

// JUnit testing
```



eclipse-workspace - Lab18/src/BalanceCheckerTest.java - Eclipse IDE

Finished after 0.087 seconds

Runs: 1/1 Errors: 0 Failures: 0

BalanceCheckerTest [Runner: JUnit 5] (

Failure Trace

```
'));
.ng and find that the stack
.nd have not yet been popped.
.is
```

273M of 557M

// **LAB 3:**

// PriorityQueuess.java

```
package priorityqueues;
import java.util.*;


public class PriorityQueuess {
```

```
public static <T> void SortArrayWithPQ(T[] toBSorted) {
PriorityQueue<T> myQueue = new PriorityQueue<T>();
for(T t : toBSorted) {
myQueue.add(t);
}
for(int i = 0; i < toBSorted.length; i++) {
toBSorted[i] = myQueue.remove();
}
}

public static <T> void printMe(T[] printy) {
for(T t : printy) {
System.out.println(t);
}
}



public static void main (String[] args) {
String[] a = {"dog", "apple", "banana", "can" ,"monkey"};
Integer[] b = {5, 7, 2, 4, 10};
Integer[] c = {50, 27, 36, 45, 81};
SortArrayWithPQ(a);
SortArrayWithPQ(b);
SortArrayWithPQ(c);

printMe(a);
printMe(b);
printMe(c);

}


}
```

Output:

```
apple
banana
can
dog
monkey
2
4
5
7
10
27
36
45
50
81
```

```
// SortTaskByPriority.java
```

```
package priorityqueues;

import java.util.Comparator;

public class SortTaskByPriority implements Comparator<Task>{


@Override
public int compare(Task arg0, Task arg1) {
return arg0.priority > arg1.priority ? 1 : -1;
}

}

// SortTaskByTime.java

package priorityqueues;

import java.util.Comparator;

public class SortTaskByTime implements Comparator<Task> {


@Override
public int compare(Task arg0, Task arg1) {
return arg0.time > arg1.time ? 1 : -1;
}

}
```

```
// Task.java

package priorityqueues;

import java.util.PriorityQueue;

public class Task {
int time;
int priority;
String description;

public Task(int time, int priority, String description) {
this.time = time;
this.priority = priority;
this.description = description;
}

@Override
public String toString() {
return "time = "  + this.time + " priority = " + this.priority + " description = " + this.description;
}
public static void main(String[] args) {
Task myTask1 = new Task(1000000,  8,  "graduate");
Task myTask2 = new Task(1440,  2, "Have a good 10/10");
Task myTask3 = new Task(20, 1, "Finish class");
Task myTask4 = new Task(7220,  3, "Computer science mid term exam");
Task myTask5 = new Task(300000000,  10,  "Get married, die");
PriorityQueue myQueueTime = new PriorityQueue<Task>(5, new SortTaskByTime());
PriorityQueue myQueuePriority = new PriorityQueue<Task>(5, new SortTaskByPriority());

myQueueTime.add(myTask1);
myQueueTime.add(myTask2);
myQueueTime.add(myTask3);
myQueueTime.add(myTask4);
myQueueTime.add(myTask5);

myQueuePriority.add(myTask1);
myQueuePriority.add(myTask2);
myQueuePriority.add(myTask3);
myQueuePriority.add(myTask4);
myQueuePriority.add(myTask5);


 for(int i = 0; i < 5; i++) {
 System.out.println("Time queue " + myQueueTime.remove().toString());
 }
 for(int i = 0; i < 5; i++) {
 System.out.println("Priority queue " + myQueuePriority.remove().toString());
 }

}
}


// OutPut:
```

```
Time queue time = 20 priority = 1 description = Finish class
Time queue time = 1440 priority = 2 description = Have a good 10/10
Time queue time = 7220 priority = 3 description = Computer science mid term exam
Time queue time = 1000000  priority = 8 description = graduate
Time queue time = 300000000  priority = 10 description = Get married, die
Priority queue time = 20 priority = 1 description = Finish class
Priority queue time = 1440  priority = 2 description = Have a good 10/10
Priority queue time = 7220  priority = 3 description = Computer science mid term exam
Priority queue time = 1000000  priority = 8 description = graduate
Priority queue time = 300000000  priority = 10 description = Get married, die
```