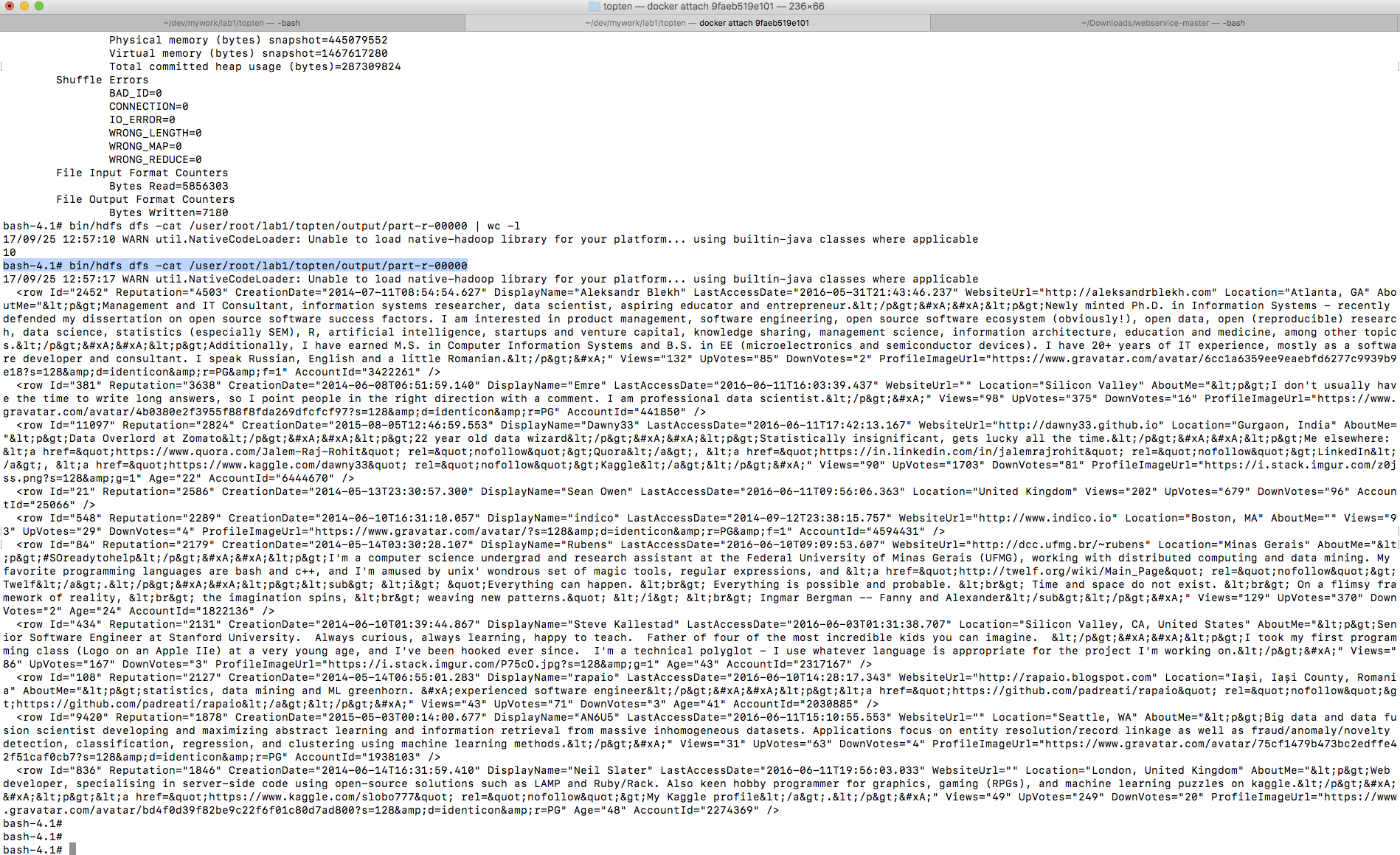
**ID2221 HT17-1 Data-Intensive Computing – MapReduce Assignment**

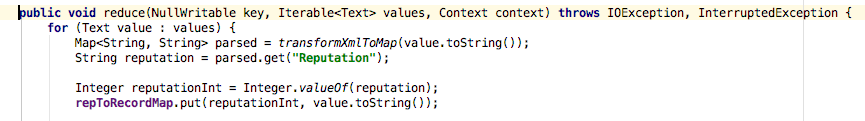
Jyoti Yadav, [jyoti@kth.se](mailto:jyoti@kth.se)

**Output:**



**Issue Faced:**

I was getting NullPointerException in reducer code for the following code, even though I added validations in the mapper code:



On adding logs I found the for some reason the records(rows) passed to the reducer were empty. I then added log in the Cleanup method of mapper code. I found that for some reason(still unknown to me) it was empty record. I then changed both the maps to:

**private** TreeMap<Integer, String> **repToRecordMap** = **new** TreeMap<>();

and converted the String to text again when writing to Context.

**Source Code:** The code below can also downloaded from my github account: **<https://github.com/jyotiyad/mapReduceAssignment>**

**package** lab1code;  
  
  
**import** org.apache.hadoop.conf.Configuration;  
**import** org.apache.hadoop.fs.Path;  
**import** org.apache.hadoop.io.NullWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Job;  
**import** org.apache.hadoop.mapreduce.Mapper;  
**import** org.apache.hadoop.mapreduce.Reducer;  
**import** org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
**import** org.apache.log4j.Logger;  
  
**import** java.io.IOException;  
**import** java.util.HashMap;  
**import** java.util.Map;  
**import** java.util.TreeMap;  
  
**public class** TopTen {  
 **private** Logger **log** = Logger.*getLogger*(TopTen.**class**);  
  
  
 **public static** Map<String, String> transformXmlToMap(String xml) {  
 Map<String, String> map = **new** HashMap<String, String>();  
 **try** {  
 String[] tokens = xml.trim().substring(5, xml.trim().length() - 3)  
 .split(**"\""**);  
  
 **for** (**int** i = 0; i < tokens.**length** - 1; i += 2) {  
 String key = tokens[i].trim();  
 String val = tokens[i + 1];  
  
 map.put(key.substring(0, key.length() - 1), val);  
 }  
 } **catch** (Exception e) {  
 System.***err***.println(xml);  
 }  
  
 **return** map;  
 }  
  
 **public static class** TopTenMapper **extends** Mapper<Object, Text, NullWritable, Text> {  
 **private** Logger **log** = Logger.*getLogger*(TopTenMapper.**class**);  
 *// Stores a map of user reputation to the record* **private** TreeMap<Integer, String> **repToRecordMap** = **new** TreeMap<>();  
  
 **public void** map(Object key, Text value, Context context) **throws** IOException, InterruptedException {  
 Map<String, String> parsed = *transformXmlToMap*(value.toString());  
 String userId = parsed.get(**"Id"**);  
 String reputation = parsed.get(**"Reputation"**);  
  
 *// check that this row contains user data* **if** (reputation == **null** || reputation.trim().length() == 0 || userId == **null** || userId.trim().length() == 0) {  
 **return**;  
 }  
  
 *//check valid reputation* Integer reputationInt = **null**;  
 **try** {  
 reputationInt = Integer.*valueOf*(reputation);  
 } **catch** (NumberFormatException e) {  
 System.***out***.println(**"skipping recrod with userId="** + userId + **" reputation="**+ reputation);  
 **log**.info(**"skipping recrod with userId="** + userId + **" reputation="**+ reputation);  
 **return**;  
 }  
  
  
 *// Add this record to our map with the reputation as the key* **repToRecordMap**.put(reputationInt, value.toString());  
  
 *// If we have more than ten records, remove the one with the lowest reputation.* **if** (**repToRecordMap**.size() > 10) {  
 Integer lowestReputation = **repToRecordMap**.firstKey();  
 **repToRecordMap**.remove(lowestReputation);  
 }  
 }  
  
 **protected void** cleanup(Context context) **throws** IOException, InterruptedException {  
 *// Output our ten records to the reducers with a null key* **for** (String t : **repToRecordMap**.values()) {  
 context.write(NullWritable.*get*(), **new** Text(t));  
 }  
 }  
 }  
  
  
 **public static class** TopTenReducer **extends** Reducer<NullWritable, Text, NullWritable, Text> {  
  
 **private** Logger **log** = Logger.*getLogger*(TopTenReducer.**class**);  
 *// Stores a map of user reputation to the record  
 // Overloads the comparator to order the reputations in descending order* **private** TreeMap<Integer, String> **repToRecordMap** = **new** TreeMap<>();  
  
 **public void** reduce(NullWritable key, Iterable<Text> values, Context context) **throws** IOException, InterruptedException {  
 **for** (Text value : values) {  
 Map<String, String> parsed = *transformXmlToMap*(value.toString());  
 String reputation = parsed.get(**"Reputation"**);  
  
 Integer reputationInt = Integer.*valueOf*(reputation);  
 **repToRecordMap**.put(reputationInt, value.toString());  
  
 *// If we have more than ten records, remove the one with the lowest reputation* **if** (**repToRecordMap**.size() > 10) {  
 Integer lowestValue= **repToRecordMap**.firstKey();  
 **repToRecordMap**.remove(lowestValue);  
 }  
 }  
  
 *// Sort in descending order* **for** (String t : **repToRecordMap**.descendingMap().values()) {  
  
 *// Output our ten records to the file system with a null key* context.write(NullWritable.*get*(), **new** Text(t));  
 }  
 }  
 }  
  
  
 **public static void** main(String[] args) **throws** Exception {  
 Configuration conf = **new** Configuration();  
 Job job = Job.*getInstance*(conf, **"top ten"**);  
 job.setNumReduceTasks(1);  
 job.setJarByClass(TopTen.**class**);  
 job.setMapperClass(TopTenMapper.**class**);  
 job.setCombinerClass(TopTenReducer.**class**);  
 job.setReducerClass(TopTenReducer.**class**);  
  
 job.setOutputKeyClass(NullWritable.**class**);  
 job.setOutputValueClass(Text.**class**);  
 FileInputFormat.*addInputPath*(job, **new** Path(args[0]));  
 FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));  
 System.*exit*(job.waitForCompletion(**true**) ? 0 : 1);  
 }  
}