Lab 2: Introduction to Serialization

Objective:

After completing this lab, you will be familiar with writing custom writables and custom comparators. Finally, the lab will also introduce you to sequence files.

Requirements:

1. Hortonworks Sandbox 2.3
2. Eclipse
3. Maven
4. M2E Plugin (A Maven Plugin for Eclipse)

Due Date: Friday 9/25 11:55 PM

Things to Include in your Submission:

Your submission should be a zip file named username_lab2.zip that comprises of 3 folders:

1. ClassSample - This folder will contain all the .java files needed to accomplish Task 1 and the .jar file that you built using Maven. The folder should also contain a text file that includes the command for executing your code on a test cluster

2. ProjectHours - This folder will contain all the .java files needed to accomplish Task 2 and the .jar file that you built using Maven. The folder should also contain a text file that includes the command for executing your code on a test cluster.

3. Poems - This folder will contain all the .java files needed to accomplish Task 3 and the .jar file that you built using Maven. The folder should also contain a text file that includes the command for executing your code on a test cluster.

4. A .pdf file that contains answers to the various questions you were asked during the lab.

Please upload the username_lab2.zip file to the Lab 2 Drop Box on Moodle

Rubric:

1. **Task 1 (20 Points)**
    a. Working Map class with correct logic - 2.5 Points
    b. Working Reduce Class with correct logic - 2.5 Points

c. Working Custom TextPair Writable - 5 Points
d. Working Custom Raw Comparator - 5 Points
e. Working Main Driver class - 2.5 Points
f. Following submission instructions - 2.5 Points

2. **Task 2 (45 Points)**
   a. Working Map class with correct logic - 5 Points
   b. Working Reduce Class with correct logic - 5 Points
   c. Working Custom Writable - 12.5 Points
   d. Working Custom Raw Comparator - 15 Points
   e. Working Main Driver class - 5 Points
   f. Following submission instructions - 2.5 Points
3. **Task 3 (20 Points)**
   a. Sequence File Writer - 10 Points
   b. Sequence File Reader - 5 Points
   c. The names of the input directory, location of the output sequence file are specified as input arguments - 3 Points
   d. All submission instructions have been followed - 2 Points
4. **Question 1 (15 Points)**. Explain the data flow of MapReduce Job in Yarn

Feedback:

Please complete the anonymous feedback for the lab under Feedback → Lab Anonymous Feedback → Lab 2 Anonymous Feedback.

1. What is serialization?

Serialization is the process of turning structured objects into a byte stream for transmission over a network or for writing to persistent storage. Deserialization is the reverse process of turning a byte stream back into a series of structured objects. Hadoop uses its own serialization format, Writables, which is compact and fast, but not so easy to extend or use from languages other than Java.

The Writable interface defines two methods: one for writing its state to a DataOutput binary stream and one for reading its state from a DataInput binary stream.

```
package org.apache.hadoop.io;
import java.io.DataOutput;
import java.io.DataInput;
import java.io.IOException;

public interface Writable {
        void write(DataOutput out) throws IOException;
```

```
    void readFields(DataInput in) throws IOException;
}
```

WritableComparable and comparators

The various hadoop specific data types (the various Writable classes - see figure in next page) implement the WritableComparable interface, which is just a subinterface of the Writable and java.lang.Comparable interfaces:

```
package org.apache.hadoop.io;
public interface WritableComparable<T> extends Writable, Comparable<T> { }
```

Comparison of types is crucial for MapReduce, where there is a sorting phase during which keys are compared with one another. One optimization that Hadoop provides is the RawComparator extension of Java's Comparator:

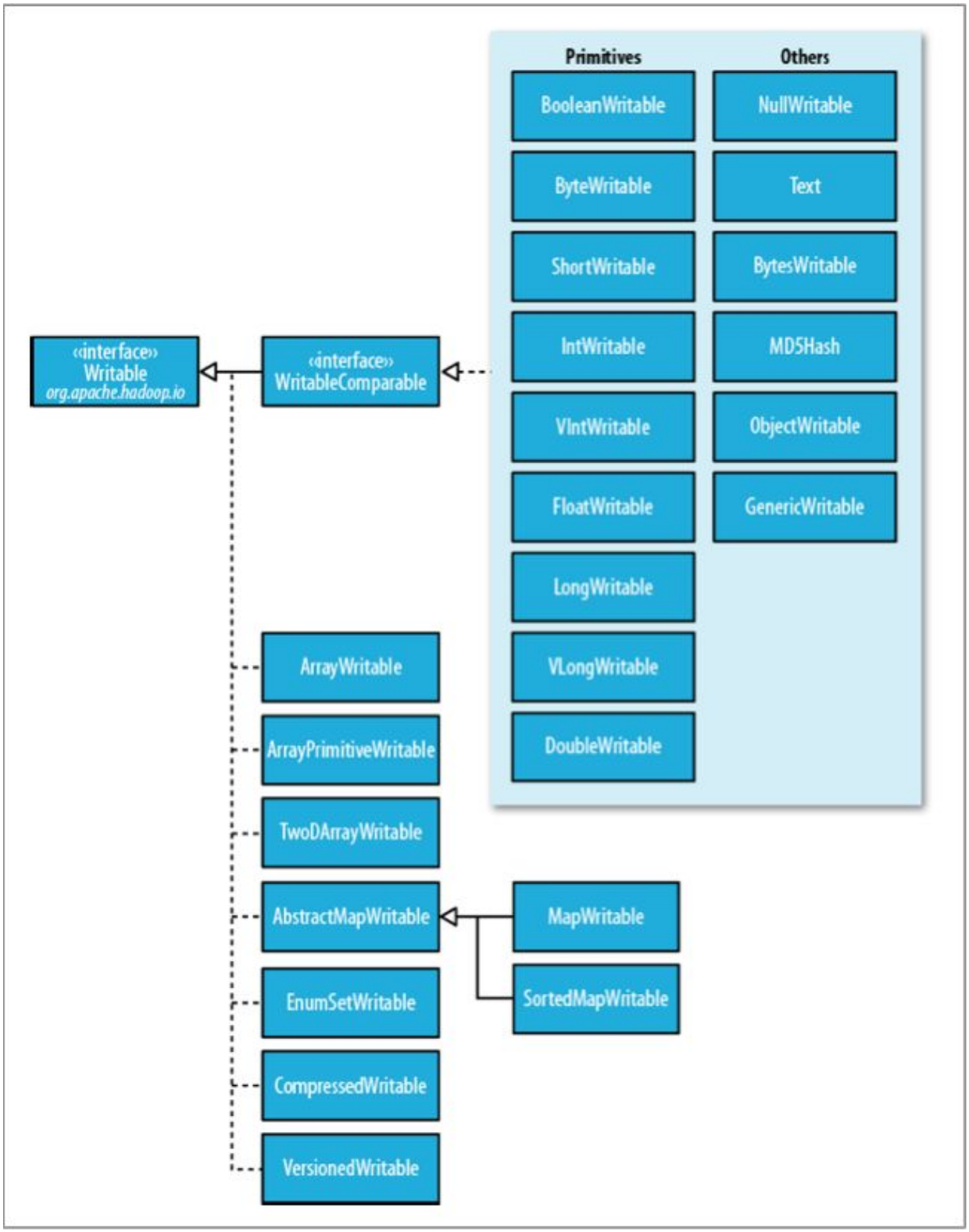
```
package org.apache.hadoop.io;
import java.util.Comparator;

public interface RawComparator<T> extends Comparator<T> {

    public int compare(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2);
}
```

This interface permits implementers to compare records read from a stream without deserializing them into objects, thereby avoiding any overhead of object creation. For example, the comparator for IntWritables implements the raw compare() method by reading an integer from each of the byte arrays b1 and b2 and comparing them directly from the given start positions (s1 and s2) and lengths (l1 and l2).

WritableComparator is a general-purpose implementation of RawComparator for WritableComparable classes. It provides two main functions. First, it provides a default implementation of the raw compare() method that deserializes the objects to be compared from the stream and invokes the object compare() method. Second, it acts as a factory for RawComparator instances (that Writable implementations have registered)



Writable Classes in Hadoop

Custom Writables in Hadoop

Hadoop comes with a useful set of Writable implementations that serve most purposes; however, on occasion, you may need to write your own custom implementation. With a custom Writable, you have full control over the binary representation and the sort order. Because Writables are at the heart of the MapReduce data path, tuning the binary representation can have a significant effect on performance. The stock Writable implementations that come with Hadoop are well-tuned, but for more elaborate structures, it is often better to create a new Writable type rather than compose the stock types.

In class, we saw an example of implementing a Custom Writable(Text Pair - See Example 4-7) and a raw comparator (TextPairComparator - Similar to Example 4-8). The code samples we used in class are available on the class home page.

Task 1: Serialization Sample (20 Points)

We will work on the dataset used during the in-class demonstration of MapReduce.

To Do 1: Recreate the program used in class to demonstrate custom writables - the program to count the number of times each person's name appeared in an input file. I would encourage you to type up the code, rather than copying and pasting it. The code used in class is available in Weekly Lecture Content under Week 2. Please be sure to run your mapreduce job with both the native comparator as well as the custom raw comparator. Please use the small dataset titled nameSample.txt as your input data set.

Turn in:

1. Create a folder called ClassSample
2. This folder will contain all the .java files needed to accomplish Task 1 and the .jar file that you built using Maven The folder should also contain a text file that includes the command for executing your code on a test cluster.

Rubric:

1) Working Map class with correct logic - 2.5 Points
2) Working Reduce Class with correct logic - 2.5 Points
3) Working Custom TextPair Writable - 5 Points
4) Working Custom Raw Comparator - 5 Points
5) Working Main Driver class - 2.5 Points
6) Following submission instructions - 2.5 Points

Task 2: Counting the number of hours worked by people on different sprints (45 Points)

Students at Rose-Hulman are used to working on sprints in their various CSSE projects. Assume that you have been asked to keep track of the hours you worked on for each sprint. Assume that the data is stored in the following format:

Sriram,Mohan,1,2.5
Sriram,Mohan,2,3.75
Sriram,Mohan,1,5.25
Dharmin,Shah,1, 2
Dharmin,Shah,2, 5.65
Dharmin,Shah,2,3.2

The above line should should be interpreted as follows: Line 1 - A person with first name sriram, last name mohan, worked on sprint 1 for 2.5 hours. Line 2 - A person with first name sriram, last name mohan, worked on sprint 2 for 3.75 hours. Line 3 - A person with first name sriram, last name mohan, worked on sprint 1 for 5.25 hours and so on.

To Do: We are going to use MapReduce to compute the number of hours a given person worked on in any sprint. Please be sure to run your mapreduce job with both the native comparator as well as the custom raw comparator. For instance, with the above 6 lines the output should look like this

Sriram Mohan 1 7.75
Sriram Mohan 2 3.75
Dharmin Shah 1 2
Dharmin Shah 2 8.85

You will have to create a custom writable that includes the user's first name, last name and sprint number to complete this task. Please use the sample dataset hoursSample.txt as your input

Turn in:

1. Create a folder called ProjectHours
2. This folder will contain all the .java files needed to accomplish Task 1 and the .jar file that you built using Maven The folder should also contain a text file that includes the command for executing your code on a test cluster.

Rubric:

1) Working Map class with correct logic - 5 Points
2) Working Reduce Class with correct logic - 5 Points
3) Working Custom Writable - 12.5 Points
4) Working Custom Raw Comparator - 15 Points
5) Working Main Driver class - 5 Points
6) Following submission instructions - 2.5 Points

Task 3 - Sequence File Demo (20 Points)

Write a simple program that reads the contents of an input directory in your local file system and combines the contents into a sequence file. The key to an entry in the sequence file should be the name of the file from which the contents are being read and the value should be one line from a file. For instance, if I had two files - poem1.txt and poem2.txt then each line in these files will generate a value and the keys will either be poem1.txt or poem2.txt depending on the file from which the line was read. The names of the input directory and location of the output sequence file should be specified as input arguments.

Verify that your sequence file writer worked by reading the contents of the sequence file by writing a java program.

Hint: You will find the sample code for sequence files from class (Examples 5-14 and 5-15) to be very useful for this task.

Turn in:

1. Create a folder called Poems.
2. This folder will contain the script(s) you wrote to accomplish Task 3.The folder should also contain a text file that includes the command for executing your code on a test cluster.

Rubric:

1. Sequence File Writer - 10 Points
2. Sequence File Reader - 5 Points
3. The names of the input directory, location of the output sequence file are specified as input arguments - 3 Points
4. All submission instructions have been followed - 2 Points

Question 1 (15 Points). Explain the data flow of a MapReduce Job in Yarn. Please don't skim on detail and the explanation should be in your own words.

Things to Include in your Submission:

Your submission should be a zip file named username_lab2.zip that comprises of 3 folders:

1. ClassExample - This folder will contain all the .java files needed to accomplish Task 1 and the .jar file that you built using Maven. The folder should also contain a text file that includes the command for executing your code on a test cluster

2. ProjectHours - This folder will contain all the .java files needed to accomplish Task 2 and the .jar file that you built using Maven. The folder should also contain a text file that includes the command for executing your code on a test cluster.

3. Poems - This folder will contain all the .java files needed to accomplish Task 3 and the .jar file that you built using Maven. The folder should also contain a text file that includes the command for executing your code on a test cluster.

4. A .pdf file that contains answers to the various questions you were asked during the lab.

Please upload the username_lab2.zip file to the Lab 2 Drop Box on Moodle

Feedback:

Please complete the anonymous feedback for the lab under Feedback → Lab Anonymous Feedback → Lab 2 Anonymous Feedback.