Caleb Post
Hadoop Lab 3 Questions and Answers.

Question 1: Explain the functionality of TextOutputFormat with an example.

TextOutputFormat is the default output format of a Hadoop Job. It functions in much the same way as the TextInputFormat. It writes out the key and the value from the reducer in string format to the output file separated by a tab.

For instance, if we have the key `Hello1` and the value `World!` that get through our reducer and output to the file through the TextOutputFormat it will look like this:

```
…
Hello1     World!
…
```

Question 2: Explain the functionality of MultipleOutputFormat with an example.

In a standard map reduce output, there is one output file per reducer that is run. There may be cases however where this is not the desired behavior. In that instance, we can use the MultipleOutputs that map reduce provides. Using MultipleOutputs, you can specify a name for the file in addition to the contents being written inside of it. Filenames from the reducer are then in the form `name-r-nnnn` and from the mapper are `name-m-nnnn`. The nnnn portion is a designated part number that ensures that output from different partitions do not collide with the same name. The name is the arbitrary portion that the programmer gets to decide upon.

An example of this can be taken from our textbook.

```
protected void reduce( Text key, Iterable < Text > values,
     Context context) throws IOException, InterruptedException {
for (Text value : values) {
     multipleOutputs.write( NullWritable.get(), value,key.toString());
} }
```

This snippet of code runs the reducer and outputs each value into a file specified by the String representation of the key.

Question 3: Explain the idea behind distributed cache with an example.

The idea behind the distributed cache is that if you have a small file which must be joined with a significantly larger file, you can upload the small file to the distributed cache and have it be made available to all machines doing the mapping and reducing. This allows much more efficient joining than was done for example in this lab.

Instead of creating multiple inputs for this lab then, we could have specified that we upload the Sprint Details text file to the distributed cache using `–files sprintDetails.txt` then, as we proceeded we would have only had to write one mapper, and we would not have needed to write the custom partitioner and grouper. Instead, when we reached the reducing step we could have accessed the file from the distributed cache using a new object which reads out the proper sprint name for the id.

```
SprintDetailsMetadata metadata = new SprintDetailsMetadata();
metadata.initialize(new File("sprintDetails.txt"));

String sprintName = metadata.getStationName(id);
```

The metadata object in this case would keep an internal array of the names read in from the file. This reduces the number of custom classes needed by at least two depending on your implementation. It also can significantly reduce processing time if the file is small enough because additional mappers are not needed to process the data.