1. Write a MapReduce program to count each unique word with case sensitivity using java.

WordCountCaseSensitive.java

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCountCaseSensitive {

  public static class TokenizerMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {

      String[] words = value.toString().split("\\s+");

      for (String w : words) {
        if (!w.isEmpty()) {
          word.set(w);
          context.write(word, one);
        }
      }
    }
  }

  public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    public void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values)
        sum += val.get();
      context.write(key, new IntWritable(sum));
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count case sensitive");
    job.setJarByClass(WordCountCaseSensitive.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

2. Write a MapReduce program to group word counts by their starting letter using java.

GroupByStartingLetter.java

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class GroupByStartingLetter {

  public static class LetterMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text letter = new Text();

    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {
      String[] words = value.toString().split("\\s+");
      for (String word : words) {
        if (!word.isEmpty()) {
          char first = word.charAt(0);
          letter.set(String.valueOf(first));
          context.write(letter, one);
        }
      }
    }
  }

  public static class LetterReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    public void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException {
      int count = 0;
      for (IntWritable val : values)
        count += val.get();
      context.write(key, new IntWritable(count));
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "group by starting letter");
    job.setJarByClass(GroupByStartingLetter.class);
    job.setMapperClass(LetterMapper.class);
    job.setReducerClass(LetterReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.setInputPaths(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
```

```
  }
}
```

3. **Write a MapReduce program to exclude common stop words ("the", "is", "and") while counting word frequency.**

WordCountWithoutStopWords.java

```
import java.io.IOException;
import java.util.Arrays;
import java.util.HashSet;
import java.util.Set;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCountWithoutStopWords {

  public static class TokenizerMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
    private static final Set<String> STOP_WORDS = new HashSet<>(Arrays.asList("the", "is", "and"));
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {
      String[] words = value.toString().toLowerCase().split("\\s+");
      for (String w : words) {
        w = w.replaceAll("[^a-z]", ""); // Remove punctuation
        if (!STOP_WORDS.contains(w) && !w.isEmpty()) {
          word.set(w);
          context.write(word, one);
        }
      }
    }
  }

  public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    public void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException {
      int count = 0;
      for (IntWritable val : values)
        count += val.get();
      context.write(key, new IntWritable(count));
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count without stop words");
    job.setJarByClass(WordCountWithoutStopWords.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setReducerClass(IntSumReducer.class);
```

```
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.setInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

TO RUN

```
hadoop fs -rm -r /user/hadoop/input

hadoop fs -mkdir -p /user/hadoop/input

hadoop fs -put input.txt /user/hadoop/input

--------------------------

nano <(class_name)>.java

javac -classpath $(hadoop classpath) -d units <(class_name)>.java

jar cf units.jar -C units/ .

hadoop jar units.jar <(class_name)> /user/hadoop/input
/user/hadoop/output

hadoop fs -cat /user/hadoop/output/part-r-00000
```