Building an Inverted Index

Building an Inverted Index

Let's say you want to build a search engine

Let's look at the steps involved at high level

Building an Inverted Index

Step 1: Build an index by crawling the web

You would download all webpages from the web and store them along with their contents

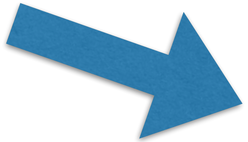
URL	Contents
abc.com	"a website"
def.com	"website"
123def.com	"just another"
..	..

Building an Inverted Index

Step 2: Build an inverted index

URL	Contents
abc.com	"a website"
def.com	"website"
123def.com	"just another"
..	..

This is an index of webpages and their contents



Build an index of words to webpages they appear in

Word	URL list
website	abc.com, def.com
a	abc.com
just	123def.com
..	

Building an Inverted Index

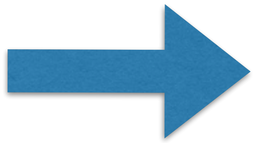
Word	URL list
website	abc.com, def.com
a	abc.com
just	123def.com
..	

Step 3: Given a search term, look up the inverted index for the relevant webpages

Building an Inverted Index

This is a classic use case for MapReduce

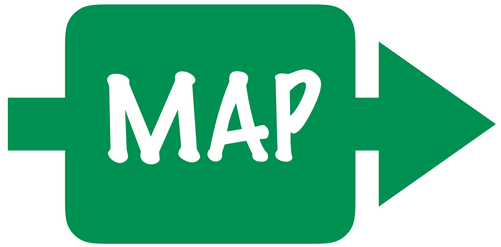
URL	Contents
abc.com	"a website"
def.com	"website"
123def.com	"just another"
..	..



Word	URL list
website	abc.com, def.com
a	abc.com
just	123def.com
..	

Building an Inverted Index

URL	Contents
abc.com	"a website"
def.com	"website"
…	…
..	..



<a, abc.com>
<website, abc.com>
<website, def.com>

For each word
in each line
in each webpage

Map output
<word, URL>

Building an Inverted Index

<a, abc.com>
<website, abc.com>
<website, def.com>

<a, abc.com>
<website, abc.com, def.com>

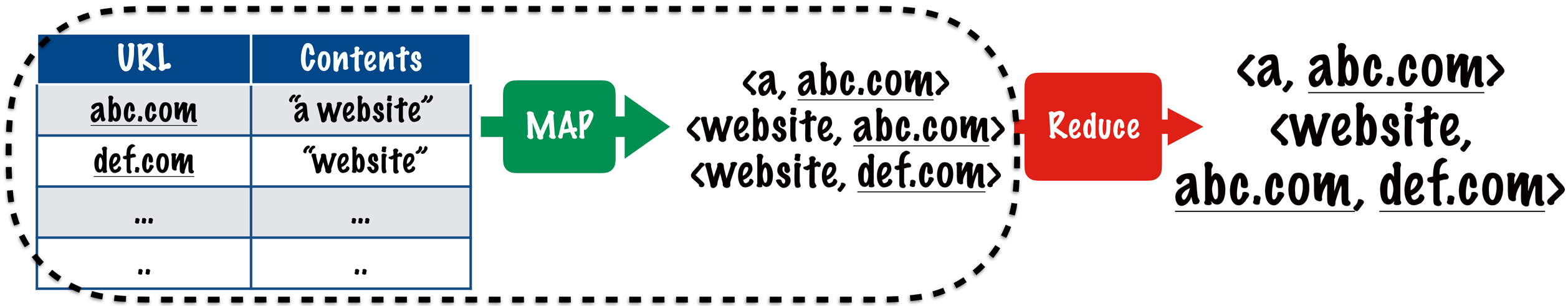
<word, URL List>

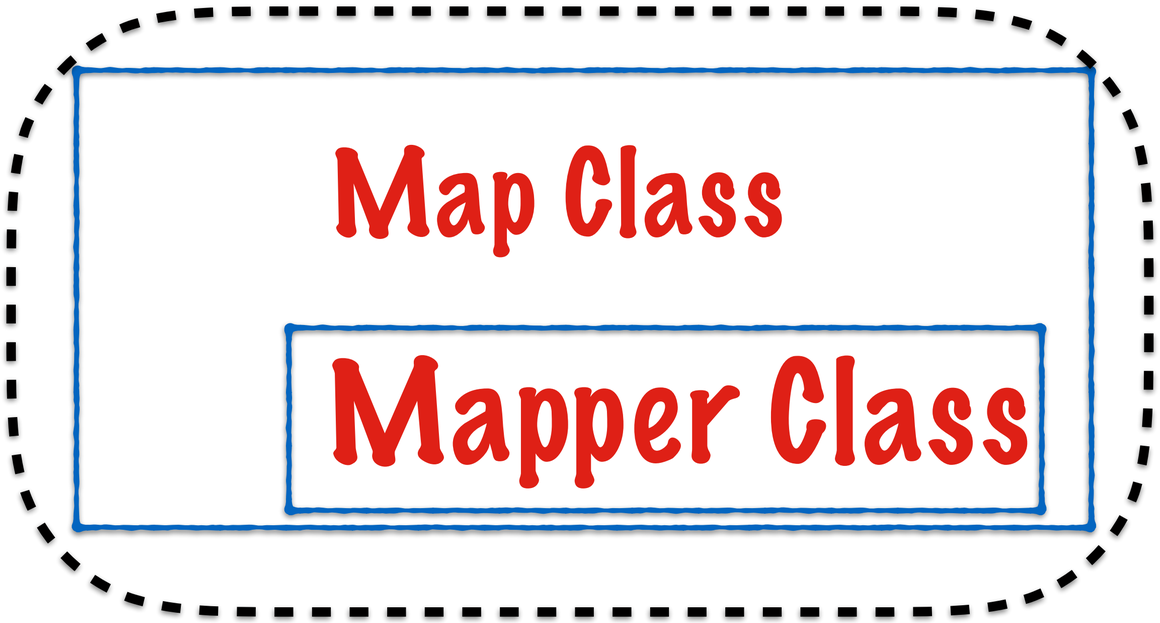
Map output
<word, URL>

Let's write the code for this in Java

Building an Inverted Index

Let's write the code for this in Java





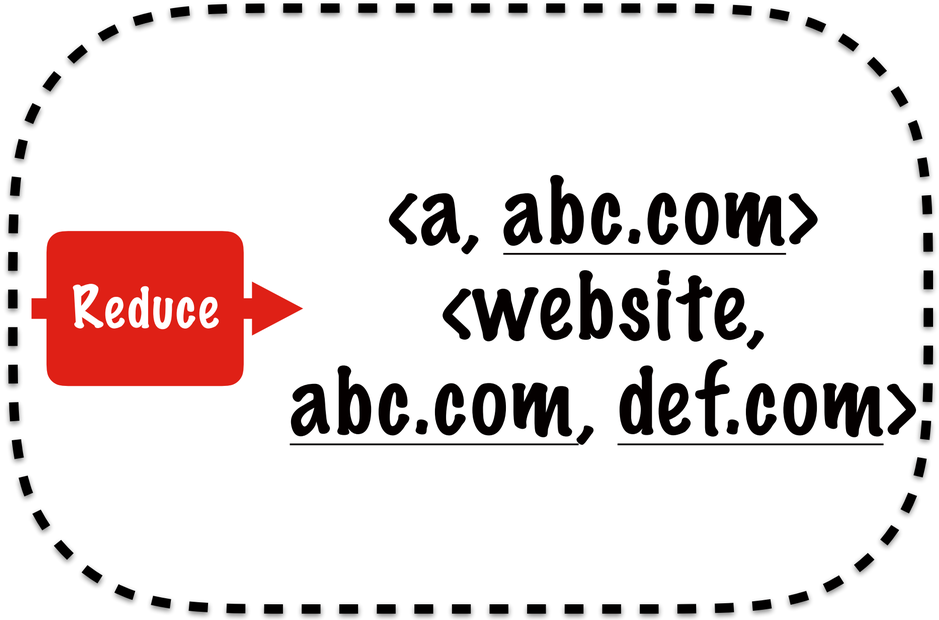
Reduce

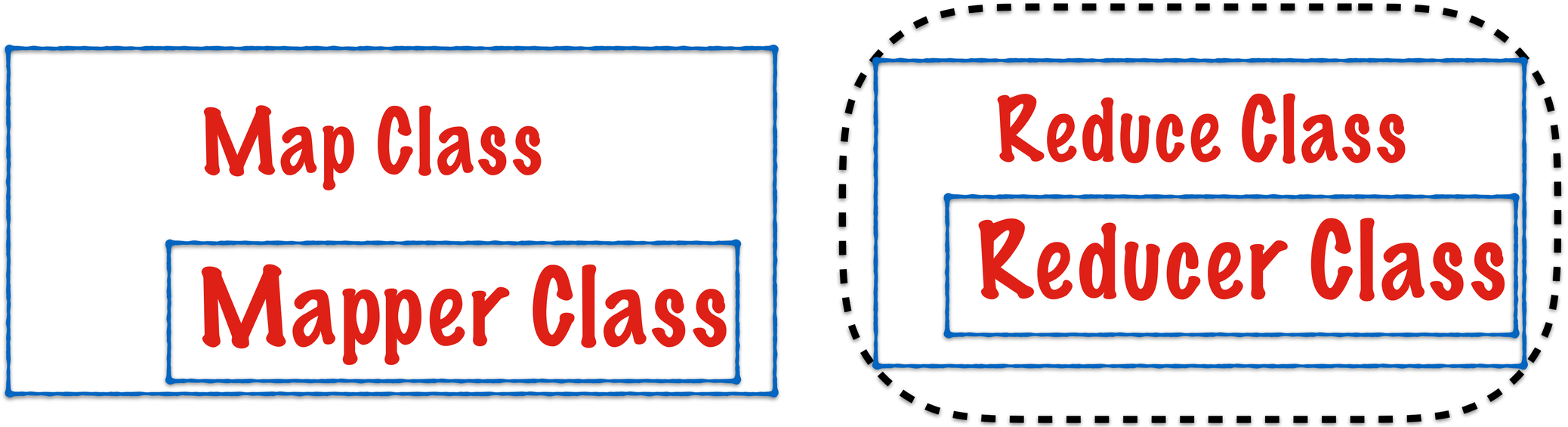
<a, abc.com>
<website, abc.com, def.com>

The map() function is implemented in a class that extends the Mapper Class

Map Class

Mapper Class





The reduce() function is implemented in a class that extends the Reducer Class

Map Class

Reduce Class

These 2 classes are used by a Job that is configured in the Main Class

invertedIndex Class

Job Object

Map Class

Reduce Class

The Job has a bunch of properties that need to be configured

invertedIndex Class

Job Object

Input filepath

Output filepath

Mapper class

Reducer class

Output data types

Map Class

Reduce Class

invertedIndex Class

Job Object

Input filepath

Mapper class

Output filepath

Reducer class

Output data types

This will be a

directory

containing all the downloaded webpages

Map Class

Reduce Class

invertedIndex Class

Job Object

Input file path

Mapper class

Output filepath

Reducer class

Output data types

The MapReduce Job will iterate through all the files present in the directory

Input path

The default input format for any MR job is text

TextInputFormat.class

Input path

This means that the input is a text file

Input path

Each line in the text file
becomes an input to map

<lineNum, line>

Input path

When you build an inverted index, the contents of each webpage are stored in 1 file

Input path

We need to process all the files in 1 MR job

Input path

When the input path is set to a directory instead of a file

Hadoop will consider all the files in the directory to be the input

Map Class

Reduce Class

invertedIndex Class

Job Object

Input file path

Mapper class

Output filepath

Reducer class

Output data types

The MapReduce Job will iterate through all the files present in the directory

Map Class

Reduce Class

Each mapper will operate on a block of data

inputSplit

Each inputSplit corresponds to 1 block from 1 file

Map Class

Reduce Class

The filename for the current input split will be the value in map's output

We'll use this to build our inverted index (instead of URL)

Map Class

Reduce Class

invertedIndex Class

Job Object

Input filepath

Output filepath

Mapper class

Reducer class

Output data types

Map Class

```java
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;
import java.util.StringTokenizer;

public class Map extends Mapper<LongWritable,Text,Text,Text> {

    private Text word = new Text();
    private Text filename = new Text();

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

            FileSplit currentSplit = ((FileSplit) context.getInputSplit());
            String filenameStr = currentSplit.getPath().getName();
            filename = new Text(filenameStr);

            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);

            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                context.write(word, filename);
            }

    }

}
```

Map Class

```
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;
import java.util.StringTokenizer;

public class Map extends Mapper<LongWritable,Text,Text,Text> {

    private Text word = new Text();
    private Text filename = new Text();

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

            FileSplit currentSplit = ((FileSplit) context.getInputSplit());
            String filenameStr = currentSplit.getPath().getName();
            filename = new Text(filenameStr);

            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);

            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                context.write(word, filename);
            }

    }

}
```

Input data types

<lineNum, line>

Map Class

```
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;
import java.util.StringTokenizer;

public class Map extends Mapper<LongWritable,Text,Text,Text> {

    private Text word = new Text();
    private Text filename = new Text();

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

            FileSplit currentSplit = ((FileSplit) context.getInputSplit());
            String filenameStr = currentSplit.getPath().getName();
            filename = new Text(filenameStr);

            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);

            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                context.write(word, filename);
            }

    }

}
```

Output data types

<word, filename>

Map Class

```
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;
import java.util.StringTokenizer;

public class Map extends Mapper<LongWritable,Text,Text,Text> {

    private Text word = new Text();
    private Text filename = new Text();


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

            FileSplit currentSplit = ((FileSplit) context.getInputSplit());
            String filenameStr = currentSplit.getPath().getName();
            filename = new Text(filenameStr);

            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);

            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                context.write(word, filename);
            }

    }

}
```

2 variables to store the map output which will be written to context

Map Class

```
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;
import java.util.StringTokenizer;

public class Map extends Mapper<LongWritable,Text,Text,Text> {

    private Text word = new Text();
    private Text filename = new Text();

      @Override
      public void map(LongWritable key, Text value, Context context) throws IOException,

            FileSplit currentSplit = ((FileSplit) context.getInputSplit());
            String filenameStr = currentSplit.getPath().getName();
            filename = new Text(filenameStr);

            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);

            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                context.write(word, filename);
            }

    }

}
```

We'll implement the map() method

Map Class

```
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;
import java.util.StringTokenizer;

public class Map extends Mapper<LongWritable,Text,Text,Text> {

    private Text word = new Text();
    private Text filename = new Text();

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

        FileSplit currentSplit = ((FileSplit) context.getInputSplit());
        String filenameStr = currentSplit.getPath().getName();
        filename = new Text(filenameStr);

            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);

            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                context.write(word, filename);
            }

    }

}
```

This section will get us the filename for the current line that's being processed

Map Class

```
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;
import java.util.StringTokenizer;

public class Map extends Mapper<LongWritable,Text,Text,Text> {

    private Text word = new Text();
    private Text filename = new Text();

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

        FileSplit currentSplit = ((FileSplit) context.getInputSplit());
        String filenameStr = currentSplit.getPath().getName();
        filename = new Text(filenameStr);

            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);

            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                context.write(word, filename);
            }

    }

}
```

This gets us the inputSplit that the mapper is currently processing

Map Class

```
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;
import java.util.StringTokenizer;

public class Map extends Mapper<LongWritable,Text,Text,Text> {

    private Text word = new Text();
    private Text filename = new Text();

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

        FileSplit currentSplit = ((FileSplit) context.getInputSplit());
        String filenameStr = currentSplit.getPath().getName();
        filename = new Text(filenameStr);

            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);

            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                context.write(word, filename);
            }

    }

}
```

FileSplit is a subclass of inputSplit

Map Class

```
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;
import java.util.StringTokenizer;

public class Map extends Mapper<LongWritable,Text,Text,Text> {

    private Text word = new Text();
    private Text filename = new Text();

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

        FileSplit currentSplit = ((FileSplit) context.getInputSplit());
        String filenameStr = currentSplit.getPath().getName();
        filename = new Text(filenameStr);

            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);

            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                context.write(word, filename);
            }

    }

}
```

FileSplit has a method that will get us the path of the file

Map Class

We get the name of the file and store it in filename

```
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;
import java.util.StringTokenizer;

public class Map extends Mapper<LongWritable,Text,Text,Text> {

    private Text word = new Text();
    private Text filename = new Text();

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

        FileSplit currentSplit = ((FileSplit) context.getInputSplit());
        String filenameStr = currentSplit.getPath().getName();
        filename = new Text(filenameStr);

            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);

            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                context.write(word, filename);
            }

    }

}
```

Map Class

```
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;
import java.util.StringTokenizer;

public class Map extends Mapper<LongWritable,Text,Text,Text> {

    private Text word = new Text();
    private Text filename = new Text();

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

            FileSplit currentSplit = ((FileSplit) context.getInputSplit());
            String filenameStr = currentSplit.getPath().getName();
            filename = new Text(filenameStr);

            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);

            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                context.write(word, filename);
            }

    }

}
```

We take the current line

Map Class

```
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;
import java.util.StringTokenizer;

public class Map extends Mapper<LongWritable,Text,Text,Text> {

    private Text word = new Text();
    private Text filename = new Text();

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

            FileSplit currentSplit = ((FileSplit) context.getInputSplit());
            String filenameStr = currentSplit.getPath().getName();
            filename = new Text(filenameStr);

            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);

            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                context.write(word, filename);
            }

    }

}
```

StringTokenizer splits the line into words

Map Class

```
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;
import java.util.StringTokenizer;

public class Map extends Mapper<LongWritable,Text,Text,Text> {

    private Text word = new Text();
    private Text filename = new Text();

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

            FileSplit currentSplit = ((FileSplit) context.getInputSplit());
            String filenameStr = currentSplit.getPath().getName();
            filename = new Text(filenameStr);

            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);

        while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                context.write(word, filename);
            }


    }

}
```

We'll iterate through all the words and write a <word,filename> pair to context

Map Class

Reduce Class

invertedIndex Class

Job Object

Input filepath

Output filepath

Mapper class

Reducer class

Output data types

Reduce Class

```java
package invertedIndex;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class Reduce extends Reducer<Text, Text,Text,Text> {
    @Override
    public void reduce(final Text key, final Iterable<Text> values,
                       final Context context) throws IOException, InterruptedException {

        StringBuilder stringBuilder = new StringBuilder();

        for (Text value : values) {
            stringBuilder.append(value.toString());

            if (values.iterator().hasNext()) {
                stringBuilder.append(" | ");
            }
        }

        context.write(key, new Text(stringBuilder.toString()));
    }
}
```

Reduce Class

```
package

import
import
import

import

public class


        StringBuilder stringBuilder =


            stringBuilder.append(value.toString());


                stringBuilder.append(
            }
        }

        context.write(key,
    }
}
```

This class will take take combine the <word, filename> pairs

Reduce Class

Input and output data types are all Text

```
package invertedIndex;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class Reduce extends Reducer<Text,Text,Text,Text> {
    @Override
    public void reduce(final Text key, final Iterable<Text> values,
                       final Context context) throws IOException, Interrupt

        StringBuilder stringBuilder = new StringBuilder();

        for (Text value : values) {
            stringBuilder.append(value.toString());

            if (values.iterator().hasNext()) {
                stringBuilder.append(" | ");
            }
        }

        context.write(key, new Text(stringBuilder.toString()));
    }
}
```

Reduce Class

We need to implement the reduce() method

```
package invertedIndex;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class Reduce extends Reducer<Text,Text,Text,Text> {
    @Override
    public void reduce(final Text key, final Iterable<Text> values,
                       final Context context) throws IOException, Interrupt

        StringBuilder stringBuilder = new StringBuilder();

        for (Text value : values) {
            stringBuilder.append(value.toString());

            if (values.iterator().hasNext()) {
                stringBuilder.append(" | ");
            }
        }

        context.write(key, new Text(stringBuilder.toString()));
    }
}
```

Reduce Class

We'll use a StringBuilder object to concatenate all the filenames

```
package invertedIndex;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class Reduce extends Reducer<Text, Text,Text,Text> {
    @Override
    public void reduce(final Text key, final Iterable<Text> values,
                       final Context context) throws IOException, InterruptedException {

        StringBuilder stringBuilder = new StringBuilder();

        for (Text value : values) {
            stringBuilder.append(value.toString());

            if (values.iterator().hasNext()) {
                stringBuilder.append(" | ");
            }
        }

        context.write(key, new Text(stringBuilder.toString()));
    }
}
```

Reduce Class

We'll iterate through the list of filenames and add each to the StringBuilder

```
package invertedIndex;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class Reduce extends Reducer<Text, Text,Text,Text> {
    @Override
    public void reduce(final Text key, final Iterable<Text> values,
                       final Context context) throws IOException, InterruptedException {

        StringBuilder stringBuilder = new StringBuilder();

        for (Text value : values) {
            stringBuilder.append(value.toString());

            if (values.iterator().hasNext()) {
                stringBuilder.append(" | ");
            }
        }

        context.write(key, new Text(stringBuilder.toString()));
    }
}
```

Reduce Class

```
package invertedIndex;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class Reduce extends Reducer<Text, Text,Text,Text> {
    @Override
    public void reduce(final Text key, final Iterable<Text> values,
                       final Context context) throws IOException, InterruptedException {

        StringBuilder stringBuilder = new StringBuilder();

        for (Text value : values) {
            stringBuilder.append(value.toString());

            if (values.iterator().hasNext()) {
                stringBuilder.append(" | ");
            }
        }

        context.write(key, new Text(stringBuilder.toString()));
    }
}
```

We are using "|" as a delimiter

Reduce Class

```
package invertedIndex;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class Reduce extends Reducer<Text, Text,Text,Text> {
    @Override
    public void reduce(final Text key, final Iterable<Text> values,
                       final Context context) throws IOException, InterruptedException {

        StringBuilder stringBuilder = new StringBuilder();

        for (Text value : values) {
            stringBuilder.append(value.toString());

            if (values.iterator().hasNext()) {
                stringBuilder.append(" | ");
            }
        }
     context.write(key, new Text(stringBuilder.toString()));
    }
}
```

We write the word and list of file names to the output

Map Class

Reduce Class

invertedIndex Class

Job Object

Input filepath

Output filepath

Mapper class

Reducer class

Output data types

invertedIndex Class

```java
package invertedIndex;

import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class invertedIndex extends Configured implements Tool{
    @Override
    public int run(String[] args) throws Exception{

        if(args.length !=2){
            System.err.println("Invalid Command");
            System.err.println("Usage: <input path> <output path>");
            return -1;
        }

        Job job = Job.getInstance(getConf());
        job.setJobName("invertedIndex");
        job.setJarByClass(invertedIndex.class);

        job.getConfiguration().set("mapreduce.output.textoutputformat.separator", " | ");

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        job.setMapperClass(Map.class);
        job.setCombinerClass(Reduce.class);
        job.setReducerClass(Reduce.class);

        Path inputFilePath = new Path(args[0]);
        Path outputFilePath = new Path(args[1]);
        FileInputFormat.addInputPath(job, inputFilePath);
        FileOutputFormat.setOutputPath(job, outputFilePath);

        FileInputFormat.setInputDirRecursive(job, true);

        return job.waitForCompletion(true) ? 0 : 1;
    }

    public static void main(String[] args) throws Exception {
        int exitCode = ToolRunner.run(new invertedIndex(), args);
        System.exit(exitCode);
    }
}
```

This class is similar to any other Main Class of an MR job

invertedIndex Class

```
package invertedIndex;

import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class invertedIndex extends Configured implements Tool{
    @Override
    public int run(String[] args) throws Exception{

        if(args.length !=2){
            System.err.println("Invalid Command");
            System.err.println("Usage: <input path> <output path>");
            return -1;
        }

        Job job = Job.getInstance(getConf());
        job.setJobName("invertedIndex");
        job.setJarByClass(invertedIndex.class);

        job.getConfiguration().set("mapreduce.output.textoutputformat.separator", " | ");

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        job.setMapperClass(Map.class);
        job.setCombinerClass(Reduce.class);
        job.setReducerClass(Reduce.class);

        Path inputFilePath = new Path(args[0]);
        Path outputFilePath = new Path(args[1]);
        FileInputFormat.addInputPath(job, inputFilePath);
        FileOutputFormat.setOutputPath(job, outputFilePath);

        FileInputFormat.setInputDirRecursive(job, true);
```

We instantiate a Job object and set the

Output data types

Mapper, Combiner, Reducer

Input and Output file paths

invertedIndex Class

```java
package invertedIndex;

import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class invertedIndex extends Configured implements Tool{
    @Override
    public int run(String[] args) throws Exception{

        if(args.length !=2){
            System.err.println("Invalid Command");
            System.err.println("Usage: <input path> <output path>");
            return -1;
        }

        Job job = Job.getInstance(getConf());
        job.setJobName("invertedIndex");
        job.setJarByClass(invertedIndex.class);

        job.getConfiguration().set("mapreduce.output.textoutputformat.separator", " | ");

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        job.setMapperClass(Map.class);
        job.setCombinerClass(Reduce.class);
        job.setReducerClass(Reduce.class);

        Path inputFilePath = new Path(args[0]);
        Path outputFilePath = new Path(args[1]);
        FileInputFormat.addInputPath(job, inputFilePath);
        FileOutputFormat.setOutputPath(job, outputFilePath);

        FileInputFormat.setInputDirRecursive(job, true);

        return job.waitForCompletion(true) ? 0 : 1;
    }

    public static void main(String[] args) throws Exception {
        int exitCode = ToolRunner.run(new invertedIndex(), args);
        System.exit(exitCode);
    }
}
```

We've also set a couple more options to control the input and output

invertedIndex Class

```
package invertedIndex;

import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class invertedIndex extends Configured implements Tool{
    @Override
    public int run(String[] args) throws Exception{

        if(args.length !=2){
            System.err.println("Invalid Command");
            System.err.println("Usage: <input path> <output path>");
            return -1;
        }

        Job job = Job.getInstance(getConf());
        job.setJobName("invertedIndex");
        job.setJarByClass(invertedIndex.class);

        job.getConfiguration().set("mapreduce.output.textoutputformat.separator", " | ");

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        job.setMapperClass(Map.class);
        job.setCombinerClass(Reduce.class);
        job.setReducerClass(Reduce.class);

        Path inputFilePath = new Path(args[0]);
        Path outputFilePath = new Path(args[1]);
        FileInputFormat.addInputPath(job, inputFilePath);
        FileOutputFormat.setOutputPath(job, outputFilePath);

        FileInputFormat.setInputDirRecursive(job, true);

        return job.waitForCompletion(true) ? 0 : 1;
    }

    public static void main(String[] args) throws Exception {
        int exitCode = ToolRunner.run(new invertedIndex(), args);
        System.exit(exitCode);
    }
}
```

The final output is normally tab separated, we are setting it to "|"

invertedIndex Class

```java
package invertedIndex;

import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class invertedIndex extends Configured implements Tool{
    @Override
    public int run(String[] args) throws Exception{

        if(args.length !=2){
            System.err.println("Invalid Command");
            System.err.println("Usage: <input path> <output path>");
            return -1;
        }

        Job job = Job.getInstance(getConf());
        job.setJobName("invertedIndex");
        job.setJarByClass(invertedIndex.class);

        job.getConfiguration().set("mapreduce.output.textoutputformat.separator", " | ");

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        job.setMapperClass(Map.class);
        job.setCombinerClass(Reduce.class);
        job.setReducerClass(Reduce.class);

        Path inputFilePath = new Path(args[0]);
        Path outputFilePath = new Path(args[1]);
        FileInputFormat.addInputPath(job, inputFilePath);
        FileOutputFormat.setOutputPath(job, outputFilePath);

        FileInputFormat.setInputDirRecursive(job, true);

        return job.waitForCompletion(true) ? 0 : 1;
    }

    public static void main(String[] args) throws Exception {
        int exitCode = ToolRunner.run(new invertedIndex(), args);
        System.exit(exitCode);
    }
}
```

This class is similar to any other Main Class of an MR job

invertedIndex Class

```
package invertedIndex;

import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class invertedIndex extends Configured implements Tool{
    @Override
    public int run(String[] args) throws Exception{

        if(args.length !=2){
            System.err.println("Invalid Command");
            System.err.println("Usage: <input path> <output path>");
            return -1;
        }

        Job job = Job.getInstance(getConf());
        job.setJobName("invertedIndex");
        job.setJarByClass(invertedIndex.class);

        job.getConfiguration().set("mapreduce.output.textoutputformat.separator", " | ");

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        job.setMapperClass(Map.class);
        job.setCombinerClass(Reduce.class);
        job.setReducerClass(Reduce.class);

        Path inputFilePath = new Path(args[0]);
        Path outputFilePath = new Path(args[1]);
        FileInputFormat.addInputPath(job, inputFilePath);
        FileOutputFormat.setOutputPath(job, outputFilePath);

        FileInputFormat.setInputDirRecursive(job, true);


        return job.waitForCompletion(true) ? 0 : 1;
    }

    public static void main(String[] args) throws Exception {
        int exitCode = ToolRunner.run(new invertedIndex(), args);
        System.exit(exitCode);
    }
}
```

By setting this property to true, the MR job will use all the files in all the subdirectories of the input directory

Not just those at the top level

Writable Interface

Writable Interface

Hadoop has a bunch of Writable classes

Text

IntWritable

LongWritable etc

Writable Interface

Text
IntWritable
LongWritable etc

These act as wrappers around the regular Java primitives

Writable Interface

Text
IntWritable
LongWritable etc

MapReduce Input and Output types are specified using these classes

Writable Interface

Text
IntWritable
LongWritable etc

They implement the
Writable Interface

Writable Interface

MapReduce input, output types
use this interface to implement methods
for optimal network serialization

Writable Interface

optimal network serialization

Serialization turns objects to byte streams

Writable Interface

optimal network serialization

These byte streams are transmitted over the network (ex: from a mapper node to a reducer node)

Remote Procedure Call (RPC) serialization

Writable Interface

The optimal network serialization process is

Compact — the byte streams are small to conserve network bandwidth

Fast — doesn't add performance overhead

Interoperable — works across multiple systems and languages

Writable Interface

```
public interface Writable {
    void write(DataOutput var1) throws IOException;

    void readFields(DataInput var1) throws IOException;
}
```

All the Writable classes implement the write() and readFields() methods

Writable Interface

The Writable classes we know actually implements both Writable and java.lang.Comparable

WritableComparable

This is because Hadoop needs to know how to sort (compare) the Writable Objects

WritableComparable

```
public interface WritableComparable<T>
extends Writable, Comparable<T> {
}
```

In addition to the readFields() and write() methods, these classes also implement compareTo()

WritableComparable

```
public interface WritableComparable<T>
extends Writable, Comparable<T> {
}
```

In addition to the readFields() and write() methods, these classes also implement compareTo()

Sometimes, you might want to create your own custom object for the MapReduce key type/value type

Implement your own custom WritableComparable

We'll see an example of this with the MR implementation for

Bigram Count

Bigram Count

Implement a Custom Writable

Bigram Count

The objective is to find the Frequency Distribution of all the Bigrams in a text file

Bigram Count

Bigrams are adjacent pairs of words

"This is a sentence"

Bigram Count

Bigrams are adjacent pairs of words

"This is a sentence"

This is

Bigram Count

Bigrams are adjacent pairs of words

"This is a sentence"

This is

is a

Bigram Count

Bigrams are **adjacent pairs** of words

This is
is a
a sentence

These are all the bigrams in this text

Bigram Count

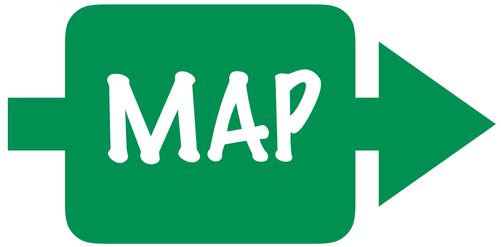
Bigrams are **adjacent pairs** of words

This is
is a
a sentence

These are all the bigrams in this text

Bigram Count

Lines
this is line 1
this is another line
…
..



<(this, is) , 1>
<(is, line), 1>
<(line,1,) 1>....

For each bigram in the text file

Map output
<bigram, 1>

Bigram Count

<(this, is) , 1>
<(is, line), 1>
<(line,1,) 1>....

<(this, is) , 2>
<(is, line), 1>
<(line,1,) 1>....

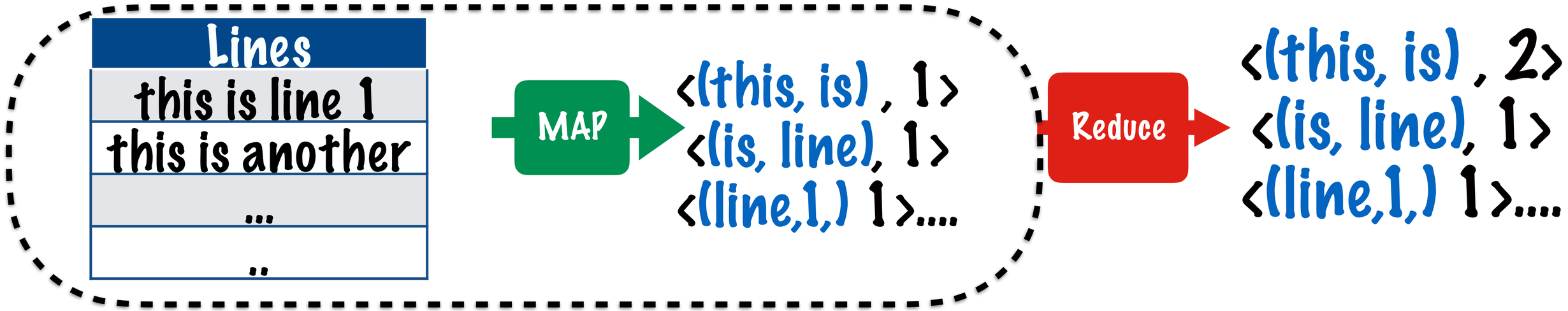
Map output <bigram, 1>

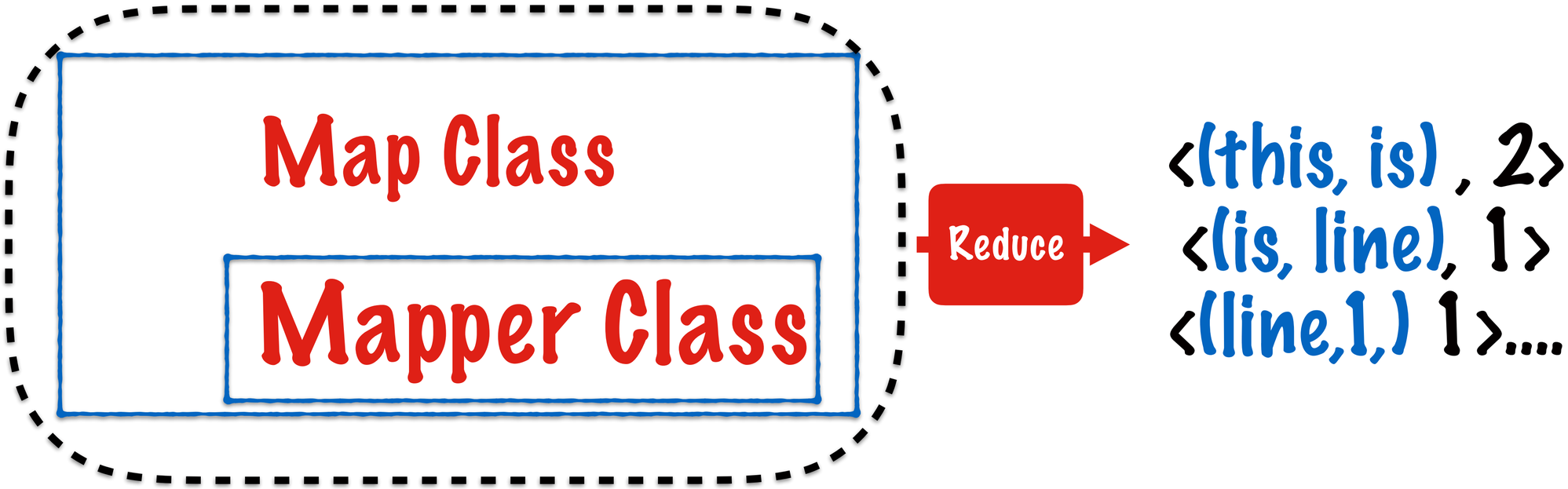
<bigram,count>

Let's write the code for this in Java

Bigram Count

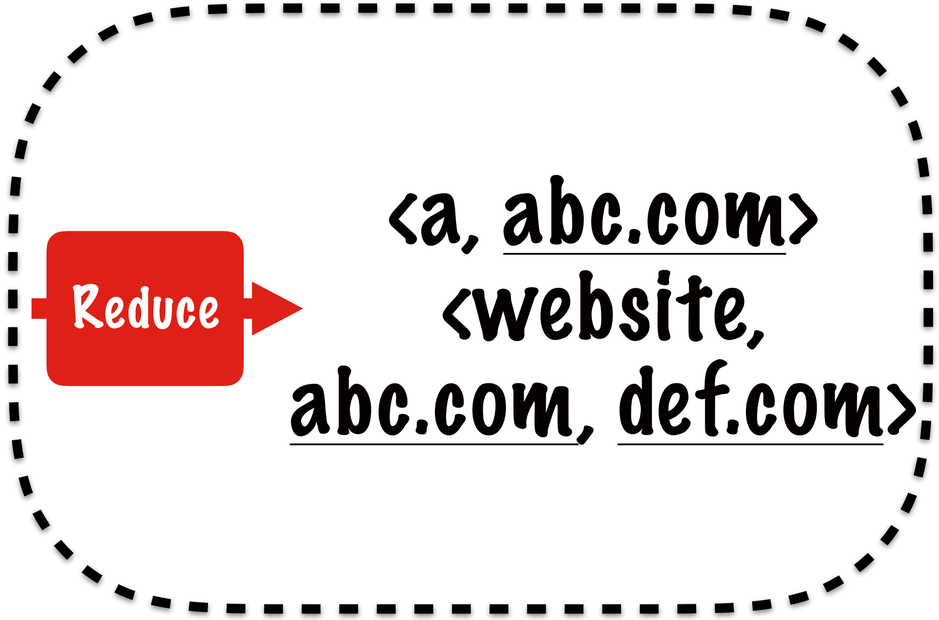
Let's write the code for this in Java




Map Class
Mapper Class
Reduce
<(this, is) , 2>
<(is, line), 1>
<(line,1,) 1>....

Map Class

Mapper Class



Map Class

Mapper Class

Reduce Class

Reducer Class

The reduce() function is implemented in a class that extends the Reducer Class

Map Class

<input key type,
input value type,
output key type,
output value type>

Mapper Class

Reduce Class

<input key type,
input value type,
output key type,
output value type>

Reducer Class

Both of these are generics for which you need to specify the input types, output types

Map Class

<input key type,
input value type,
output key type,
output value type>

Reduce Class

<input key type,
input value type,
output key type,
output value type>

Mapper Class

Reducer Class

All of these implement the

WritableComparable Interface

Map Class

<input key type,
input value type,
output key type,
output value type>

Reduce Class

<input key type,
input value type,
output key type,
output value type>

The map output key is a Bigram

A Bigram is a pair of words

Map Class

<input key type,
input value type,
output key type,
output value type>

Reduce Class

<input key type,
input value type,
output key type,
output value type>

Mapper Class

Reducer Class

We can implement a custom WritableComparable class

to represent Bigrams

Map Class

<input key type,
input value type,
TextPair,
output value type>

Reduce Class

<TextPair,
input value type,
output key type,
output value type>

Mapper Class

Reducer Class

TextPair Class

WritableComparable Class

Map Class

Mapper Class

Reduce Class

Reducer Class

TextPair Class

WritableComparable Class

We'll have our Main Class as usual

Map Class

Mapper Class

Reduce Class

Reducer Class

TextPair Class

WritableComparable Class

Bigram Class

TextPair Class

```
public class TextPair implements WritableComparable<TextPair>{

    private Text first;
    private Text second;

    public void set(Text first, Text second){
        this.first=first;
        this.second=second;
    }

    public Text getFirst() {
        return first;
    }
    public Text getSecond() {
        return second;
    }


    public TextPair(){
        set(new Text(),new Text());
    }

    public TextPair(String first, String second){
        set(new Text(first),new Text(second));
    }

    public TextPair(Text first, Text second){
        set(first,second);
    }

    @Override
    public void write(DataOutput out) throws IOException{
        first.write(out);
        second.write(out);
    }

    @Override
    public void readFields(DataInput in) throws IOException{
        first.readFields(in);
        second.readFields(in);
    }

    @Override
    public int compareTo(TextPair tp){
        int cmp = first.compareTo(tp.first);
        if (cmp!=0) {
            return cmp;
```

TextPair Class

```
public class TextPair implements
WritableComparable<TextPair>{

    private Text first;
    private Text second;

    public void set(Text first, Text second){
        this.first=first;
        this.second=second;
    }

    public Text getFirst() {
        return first;
    }
    public Text getSecond() {
        return second;
    }


    public TextPair(){
        set(new Text(),new Text());
    }

    public TextPair(String first, String second){
        set(new Text(first),new Text(second));
    }

    public TextPair(Text first, Text second){
        set(first,second);
    }

    @Override
    public void write(DataOutput out) throws IOException{
        first.write(out);
        second.write(out);
    }

    @Override
    public void readFields(DataInput in) throws IOException{
        first.readFields(in);
        second.readFields(in);
    }

    @Override
    public int compareTo(TextPair tp){
```

This class implements WritableComparable

TextPair Class

```
public class TextPair implements WritableComparable<TextPair>{

    private Text first;
    private Text second;

    public void set(Text first, Text second){
        this.first=first;
        this.second=second;
    }

    public Text getFirst() {
        return first;
    }
    public Text getSecond() {
        return second;
    }


    public TextPair(){
        set(new Text(),new Text());
    }

    public TextPair(String first, String second){
        set(new Text(first),new Text(second));
    }

    public TextPair(Text first, Text second){
        set(first,second);
    }

    @Override
    public void write(DataOutput out) throws IOException{
        first.write(out);
        second.write(out);
    }

    @Override
    public void readFields(DataInput in) throws IOException{
        first.readFields(in);
        second.readFields(in);
    }

    @Override
    public int compareTo(TextPair tp){
        int cmp = first.compareTo(tp.first);
        if (cmp!=0) {
            return cmp;
```

The class represents a Bigram

A Bigram is a pair of words
Each word can be
represented by a Text object

Text is the Hadoop
wrapper for Java String

TextPair Class

```
public class TextPair implements WritableComparable<TextPair>{

    private Text first;
    private Text second;

    public void set(Text first, Text second){
        this.first=first;
        this.second=second;
    }

    public Text getFirst() {
        return first;
    }
    public Text getSecond() {
        return second;
    }


    public TextPair(){
        set(new Text(),new Text());
    }

    public TextPair(String first, String second){
        set(new Text(first),new Text(second));
    }

    public TextPair(Text first, Text second){
        set(first,second);
    }

    @Override
    public void write(DataOutput out) throws IOException{
        first.write(out);
        second.write(out);
    }

    @Override
    public void readFields(DataInput in) throws IOException{
        first.readFields(in);
        second.readFields(in);
    }

    @Override
```

These 2 objects represent the first and second elements of the Bigram

TextPair Class

```
public class TextPair implements WritableComparable<TextPair>{

    private Text first;
    private Text second;


        public void set(Text first, Text second)
{
            this.first=first;
            this.second=second;
        }

        public Text getFirst() {
            return first;
        }
        public Text getSecond() {
            return second;
        }



        public TextPair(){
            set(new Text(),new Text());
        }
    public TextPair(String first, String second){
        set(new Text(first),new Text(second));
    }
    public TextPair(Text first, Text second){
        set(first,second);
    }
    @Override
```

We need getters and setters for these 2 variables

TextPair Class

The TextPair can be instantiated in multiple ways

Without any words

```
public class TextPair implements WritableComparable<TextPair>{
    private Text first;
    private Text second;

    public void set(Text first, Text second){
        this.first=first;
        this.second=second;
    }

    public Text getFirst() {
        return first;
    }
    public Text getSecond() {
        return second;
    }

    public TextPair(){
        set(new Text(),new Text());
    }

    public TextPair(String first, String second){
        set(new Text(first),new Text(second));
    }

    public TextPair(Text first, Text second){
        set(first,second);
    }

    @Override
    public void write(DataOutput out) throws IOException{
        first.write(out);
        second.write(out);
    }

    @Override
    public void readFields(DataInput in) throws IOException{
        first.readFields(in);
        second.readFields(in);
    }

    @Override
```

TextPair Class

The TextPair can be instantiated in multiple ways

Using a pair of strings

```
public class TextPair implements WritableComparable<TextPair>{
    private Text first;
    private Text second;

    public void set(Text first, Text second){
        this.first=first;
        this.second=second;
    }

    public Text getFirst() {
        return first;
    }
    public Text getSecond() {
        return second;
    }

        public TextPair(){
            set(new Text(),new Text());
        }

        public TextPair(String first, String second){
            set(new Text(first),new Text(second));
        }

        public TextPair(Text first, Text second){
            set(first,second);
        }

    @Override
    public void write(DataOutput out) throws IOException{
        first.write(out);
        second.write(out);
    }

    @Override
    public void readFields(DataInput in) throws IOException{
        first.readFields(in);
        second.readFields(in);
    }

    @Override
```

TextPair Class

The TextPair can be instantiated in multiple ways

Using a pair of Text objects

```
public class TextPair implements WritableComparable<TextPair>{
    private Text first;
    private Text second;

    public void set(Text first, Text second){
        this.first=first;
        this.second=second;
    }

    public Text getFirst() {
        return first;
    }
    public Text getSecond() {
        return second;
    }

    public TextPair(){
        set(new Text(),new Text());
    }

    public TextPair(String first, String second){
        set(new Text(first),new Text(second));
    }

    public TextPair(Text first, Text second){
        set(first,second);
    }

    @Override
    public void write(DataOutput out) throws IOException{
        first.write(out);
        second.write(out);
    }

    @Override
    public void readFields(DataInput in) throws IOException{
        first.readFields(in);
        second.readFields(in);
    }

    @Override
```

TextPair Class

```
public class TextPair implements WritableComparable<TextPair>{

    private Text first;
    private Text second;

    public void set(Text first, Text second){
        this.first=first;
        this.second=second;
    }

    public Text getFirst() {
        return first;
    }
    public Text getSecond() {
        return second;
    }


    public TextPair(){
        set(new Text(),new Text());
    }

    public TextPair(String first, String second){
        set(new Text(first),new Text(second));
    }

    public TextPair(Text first, Text second){
        set(first,second);
    }

    @Override
    public void write(DataOutput out) throws IOException{
        first.write(out);
        second.write(out);
    }

    @Override
    public void readFields(DataInput in) throws IOException{
        first.readFields(in);
        second.readFields(in);
    }

    @Override
    public int compareTo(TextPair tp){
        int cmp = first.compareTo(tp.first);
        if (cmp!=0) {
            return cmp;
        }
        return second.compareTo(tp.second);
```

Any WritableComparable needs to implement write(), readFields(), compareTo() methods

TextPair Class

write(), readFields()

```
}
public Text getSecond() {
    return second;
}

public TextPair(){
    set(new Text(),new Text());
}

public TextPair(String first, String second){
    set(new Text(first),new Text(second));
}

public TextPair(Text first, Text second){
    set(first,second);
}
```

```
    @Override
    public void write(DataOutput out) throws IOException{
        first.write(out);
        second.write(out);
    }

    @Override
    public void readFields(DataInput in) throws IOException{
        first.readFields(in);
        second.readFields(in);
    }
```

We can use the write() and readFields() methods of the Text class

```
@Override
public int compareTo(TextPair tp){
    int cmp = first.compareTo(tp.first);
    if (cmp!=0) {
        return cmp;
    }
    return second.compareTo(tp.second);
}

@Override
public int hashCode(){
    return first.hashCode()*163 + second.hashCode();
}

@Override
public boolean equals(Object o){
    if( o instanceof TextPair){
        TextPair tp = (TextPair) o;
        return first.equals(tp.first) && second.equals(tp.second);
    }
    return  false;
}

@Override
public String toString(){
```

TextPair Class

compareTo()

TextPairs are sorted by first word and then second word

```
}
public Text getSecond() {
    return second;
}

public TextPair(){
    set(new Text(),new Text());
}
public TextPair(String first, String second){
    set(new Text(first),new Text(second));
}

public TextPair(Text first, Text second){
    set(first,second);
}

@Override
public void write(DataOutput out) throws IOException{
    first.write(out);
    second.write(out);
}

@Override
public void readFields(DataInput in) throws IOException{
    first.readFields(in);
    second.readFields(in);
}
```

```
@Override
public int compareTo(TextPair tp){
    int cmp = first.compareTo(tp.first);
    if (cmp!=0) {
        return cmp;
    }
    return second.compareTo(tp.second);
}
```

```
@Override
public int hashCode(){
    return first.hashCode()*163 + second.hashCode();
}

@Override
public boolean equals(Object o){
    if( o instanceof TextPair){
        TextPair tp = (TextPair) o;
        return first.equals(tp.first) && second.equals(tp.second);
    }
    return  false;
}

@Override
public String toString(){
```

TextPair Class

```
public class TextPair implements WritableComparable<TextPair>{

    private Text first;
    private Text second;

    public void set(Text first, Text second){
        this.first=first;
        this.second=second;
    }

    public Text getFirst() {
        return first;
    }
    public Text getSecond() {
        return second;
    }


    public TextPair(){
        set(new Text(),new Text());
    }

    public TextPair(String first, String second){
        set(new Text(first),new Text(second));
    }

    public TextPair(Text first, Text second){
        set(first,second);
    }

    @Override
    public void write(DataOutput out) throws IOException{
        first.write(out);
        second.write(out);
    }

    @Override
    public void readFields(DataInput in) throws IOException{
        first.readFields(in);
        second.readFields(in);
    }

    @Override
    public int compareTo(TextPair tp){
        int cmp = first.compareTo(tp.first);
        if (cmp!=0) {
            return cmp;
        }
        return second.compareTo(tp.second);
```

There are a few other methods we'll implement

```
    first.readFields(in);
    second.readFields(in);
}

@Override
public int compareTo(TextPair tp){
    int cmp = first.compareTo(tp.first);
    if (cmp!=0) {
        return cmp;
    }
    return second.compareTo(tp.second);
}
```

TextPair Class

```
@Override
public int hashCode(){
    return first.hashCode()*163 + second.hashCode();
}
```

```
@Override
public boolean equals(Object o){
    if( o instanceof TextPair){
        TextPair tp = (TextPair) o;
        return first.equals(tp.first) && second.equals(tp.second);
    }
    return  false;
}

@Override
public String toString(){
    return first +"\t" +second;
}
```

This is used to assign a Partition id when there are multiple reducers

```
    first.readFields(in);
    second.readFields(in);
}

@Override
public int compareTo(TextPair tp){
    int cmp = first.compareTo(tp.first);
    if (cmp!=0) {
        return cmp;
    }
    return second.compareTo(tp.second);
}
```

TextPair Class

```
@Override
public int hashCode(){
    return first.hashCode()*163 + second.hashCode();
}
```

```
@Override
public boolean equals(Object o){
    if( o instanceof TextPair){
        TextPair tp = (TextPair) o;
        return first.equals(tp.first) && second.equals(tp.second);
    }
    return  false;
}

@Override
public String toString(){
    return first +"\t" +second;
}
```

This is used to assign a Partition id when there are multiple reducers

TextPair Class

```
        second.write(out);
    }

    @Override
    public void readFields(DataInput in) throws IOException{
        first.readFields(in);
        second.readFields(in);
    }

    @Override
    public int compareTo(TextPair tp){
        int cmp = first.compareTo(tp.first);
        if (cmp!=0) {
            return cmp;
        }
        return second.compareTo(tp.second);
    }

    @Override
    public int hashCode(){
        return first.hashCode()*163 + second.hashCode();
    }
```

```
    @Override
    public boolean equals(Object o){
        if( o instanceof TextPair){
            TextPair tp = (TextPair) o;
            return first.equals(tp.first) && second.equals(tp.second);
        }
        return  false;
    }
```

```
    @Override
    public String toString(){
        return first +"\t" +second;

    }

}
```

The equals() method is also needed for sorting TextPair objects

TextPair Class

```
    @Override
    public void write(DataOutput out) throws IOException{
        first.write(out);
        second.write(out);
    }

    @Override
    public void readFields(DataInput in) throws IOException{
        first.readFields(in);
        second.readFields(in);
    }

    @Override
    public int compareTo(TextPair tp){
        int cmp = first.compareTo(tp.first);
        if (cmp!=0) {
            return cmp;
        }
        return second.compareTo(tp.second);
    }


    @Override
    public int hashCode(){
        return first.hashCode()*163 + second.hashCode();
    }

    @Override
    public boolean equals(Object o){
        if( o instanceof TextPair){
            TextPair tp = (TextPair) o;
            return first.equals(tp.first) && second.equals(tp.second);
        }
        return  false;


    }
```

The toString() method is used to generate text that will be written to the final output file

```
    @Override
    public String toString(){
        return first +"\t" +second;


    }



}
```

Map Class

Mapper Class

Reduce Class

Reducer Class

TextPair Class

WritableComparable Class

Bigram Class

Map Class

Mapper Class

Reduce Class

Reducer Class

TextPair Class

WritableComparable Class

Bigram Class

Bigram Class

```java
public class bigram  extends Configured implements Tool{

    @Override
    public int run(String[] args) throws Exception{

        if(args.length !=2){
            System.err.println("Invalid Command");
            System.err.println("Usage: Bigram <input path> <output path>");
            return -1;
        }
        Configuration conf = new Configuration();

        Job job = new Job(conf, "wordcount");


        job.setJarByClass(getClass());

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.setMapperClass(Map.class);
        job.setReducerClass(Reduce.class);

        job.setMapOutputKeyClass(TextPair.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        return job.waitForCompletion(true)?0:1;
    }

    public static void main(String[] args) throws Exception {
        int exitCode = ToolRunner.run(new bigram(), args);
        System.exit(exitCode);
    }

}
```

This is a standard Main Class

Bigram Class

```
public class bigram  extends Configured implements Tool{

    @Override
    public int run(String[] args) throws Exception{

        if(args.length !=2){
            System.err.println("Invalid Command");
            System.err.println("Usage: Bigram <input path> <output path>");
            return -1;
        }
        Configuration conf = new Configuration();

        Job job = new Job(conf, "wordcount");


        job.setJarByClass(getClass());

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.setMapperClass(Map.class);
        job.setReducerClass(Reduce.class);

        job.setMapOutputKeyClass(TextPair.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        return job.waitForCompletion(true)?0:1;
    }

    public static void main(String[] args) throws Exception {
        int exitCode = ToolRunner.run(new bigram(), args);
        System.exit(exitCode);
    }

}
```

We'll set the map output key type to TextPair

Map Class

Mapper Class

Reduce Class

Reducer Class

TextPair Class

WritableComparable Class

Bigram Class

Map Class

Mapper Class

Reduce Class

Reducer Class

TextPair Class

WritableComparable Class

Bigram Class

Map Class

```
public class Map extends Mapper<LongWritable,Text,TextPair,IntWritable> {
    private  Text lastWord = null;
    private  Text currentWord = new Text();


    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException
    {
        String line = value.toString();
        line = line.replace(",", "");
        line = line.replace(".", "");

        for(String word: line.split(" "))
        {
            if(lastWord == null)
            {
                lastWord = new Text(word);
            }
            else
            {
                currentWord.set(word);
                context.write(new TextPair(lastWord, currentWord), new IntWritable(1));
                lastWord.set(currentWord.toString());
            }
        }
    }
}
```

Map Class

```
public class Map extends
Mapper<LongWritable,Text,TextPair,IntWritable> {
    private  Text lastWord = null;
    private  Text currentWord = new Text();


    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException
    {
        String line = value.toString();
        line = line.replace(",", "");
        line = line.replace(".", "");

        for(String word: line.split(" "))
        {
            if(lastWord == null)
            {
                lastWord = new Text(word);
            }
            else
            {
                currentWord.set(word);
                context.write(new TextPair(lastWord, currentWord), new IntWritable(1));
                lastWord.set(currentWord.toString());
            }
        }
    }
}
```

Here too, we'll set the map output key type to TextPair

Map Class

```
public class Map extends Mapper<LongWritable,Text,TextPair,IntWritable> {
    private  Text lastWord = null;
     private  Text currentWord = new Text();


    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException
    {
        String line = value.toString();
        line = line.replace(",", "");
        line = line.replace(".", "");

        for(String word: line.split(" "))
        {
            if(lastWord == null)
            {
                lastWord = new Text(word);
            }
            else
            {
                currentWord.set(word);
                context.write(new TextPair(lastWord, currentWord), new IntWritable(1));
                lastWord.set(currentWord.toString());
            }
        }
    }
}
```

As we iterate through the words in a line, we'll keep track of the last word and current word

Map Class

```
public class Map extends Mapper<LongWritable,Text,TextPair,IntWritable> {
    private  Text lastWord = null;
    private  Text currentWord = new Text();


    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException
    {
        String line = value.toString();
        line = line.replace(",", "");
        line = line.replace(".", "");

        for(String word: line.split(" "))
        {
            if(lastWord == null)
            {
                lastWord = new Text(word);
            }
            else
            {
                currentWord.set(word);
                context.write(new TextPair(lastWord, currentWord), new IntWritable(1));
                lastWord.set(currentWord.toString());
            }
        }
    }
}
```

Here we implement the map()

Map Class

```
public class Map extends Mapper<LongWritable,Text,TextPair,IntWritable> {
    private  Text lastWord = null;
    private  Text currentWord = new Text();


    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException
    {
        String line = value.toString();
         line = line.replace(",", "");
         line = line.replace(".", "");

        for(String word: line.split(" "))
        {
            if(lastWord == null)
            {
                lastWord = new Text(word);
            }
            else
            {
                currentWord.set(word);
                context.write(new TextPair(lastWord, currentWord), new IntWritable(1));
                lastWord.set(currentWord.toString());
            }
        }
    }
}
```

Some cleanup so we don't get Bigrams in which 1 element is a punctuation mark

Map Class

```
public class Map extends Mapper<LongWritable,Text,TextPair,IntWritable> {
    private  Text lastWord = null;
    private  Text currentWord = new Text();


    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException
    {
        String line = value.toString();
        line = line.replace(",", "");
        line = line.replace(".", "");

        for(String word: line.split(" "))
            {
            if(lastWord == null)
            {
                lastWord = new Text(word);
            }
            else
            {
                currentWord.set(word);
                context.write(new TextPair(lastWord, currentWord), new IntWritable(1));
                lastWord.set(currentWord.toString());
            }
        }
    }
}
```

We iterate through each word

Map Class

```
public class Map extends Mapper<LongWritable,Text,TextPair,IntWritable> {
    private  Text lastWord = null;
    private  Text currentWord = new Text();


    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException
    {
        String line = value.toString();
        line = line.replace(",", "");
        line = line.replace(".", "");

        for(String word: line.split(" "))
        {
                    if(lastWord == null)
                    {
                        lastWord = new Text(word);
                    }
            else
            {
                currentWord.set(word);
                context.write(new TextPair(lastWord, currentWord), new IntWritable(1));
                lastWord.set(currentWord.toString());
            }
        }
    }
}
```

If we are at the first word in a line we don't write any output

Map Class

```
public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException
{
    String line = value.toString();
    line = line.replace(",", "");
    line = line.replace(".", "");

    for(String word: line.split(" "))
    {
        if(lastWord == null)
        {
            lastWord = new Text(word);
        }
        else
            {
                currentWord.set(word);
                context.write(new TextPair(lastWord, currentWord), new IntWritable(1));
                lastWord.set(currentWord.toString());
            }
    }
}
```

Else, the output key is a TextPair made up of the last word and the current word

Map Class

```
public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException
{
    String line = value.toString();
    line = line.replace(",", "");
    line = line.replace(".", "");

    for(String word: line.split(" "))
    {
        if(lastWord == null)
        {
            lastWord = new Text(word);
        }
        else
            {
                currentWord.set(word);
                context.write(new TextPair(lastWord, currentWord), new IntWritable(1));
                lastWord.set(currentWord.toString());
            }
    }
}
```

The output value is just 1

Map Class

```
public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException
{
    String line = value.toString();
    line = line.replace(",", "");
    line = line.replace(".", "");

    for(String word: line.split(" "))
    {
        if(lastWord == null)
        {
            lastWord = new Text(word);
        }
        else
            {
                currentWord.set(word);
                context.write(new TextPair(lastWord, currentWord), new IntWritable(1));
                lastWord.set(currentWord.toString());
            }
    }
}
```

Before we move on, we make the current word the last word

Map Class

Mapper Class

Reduce Class

Reducer Class

TextPair Class

WritableComparable Class

Bigram Class

Map Class

Mapper Class

Reduce Class

Reducer Class

TextPair Class

WritableComparable Class

Bigram Class

Reduce Class

```
public class Reduce extends Reducer<TextPair, IntWritable, Text,
IntWritable>{
    @Override
    public void reduce(TextPair key,  Iterable<IntWritable>
values, Context context) throws IOException, InterruptedException
    {
        int count=0;
        for(IntWritable value: values)
        {
            count += value.get();
        }

        context.write(new Text(key.toString()), new
IntWritable(count));
    }
}
```

Reduce Class

```
public class Reduce extends Reducer<TextPair,
IntWritable, Text, IntWritable>{

     @Override
    public void reduce(TextPair key,  Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException
    {
        int count=0;
        for(IntWritable value: values)
        {
            count += value.get();
        }

        context.write(new Text(key.toString()), new IntWritable(count));
    }
}
```

The Reduce input key type is a TextPair

Reduce Class

```
public class Reduce extends Reducer<TextPair, IntWritable, Text, IntWritable>{
    @Override
    public void reduce(TextPair key,  Iterable<IntWritable> values, Context context)
throws IOException, InterruptedException
    {
        int count=0;
           for(IntWritable value: values)
           {
                count += value.get();
           }

        context.write(new Text(key.toString()), new IntWritable(count));
    }
}
```

In reduce(), we simply compute the count for each Input Key

Reduce Class

```
ublic class Reduce extends Reducer<TextPair, IntWritable, Text, IntWritable>{
    @Override
    public void reduce(TextPair key,  Iterable<IntWritable> values, Context context) throws
OException, InterruptedException
    {
        int count=0;
        for(IntWritable value: values)
        {
            count += value.get();
        }

        context.write(new Text(key.toString()), new IntWritable(count));
    }
```

We'll write out the Bigram (as text) and the count

Reduce Class

```
ublic class Reduce extends Reducer<TextPair, IntWritable, Text, IntWritable>{
    @Override
    public void reduce(TextPair key,  Iterable<IntWritable> values, Context context) throws
OException, InterruptedException
    {
        int count=0;
        for(IntWritable value: values)
        {
            count += value.get();
        }

        context.write(new Text(key.toString()), new IntWritable(count));
    }
```

We're converting the TextPair to Text so that it can be written to an output text file

Map Class

Mapper Class

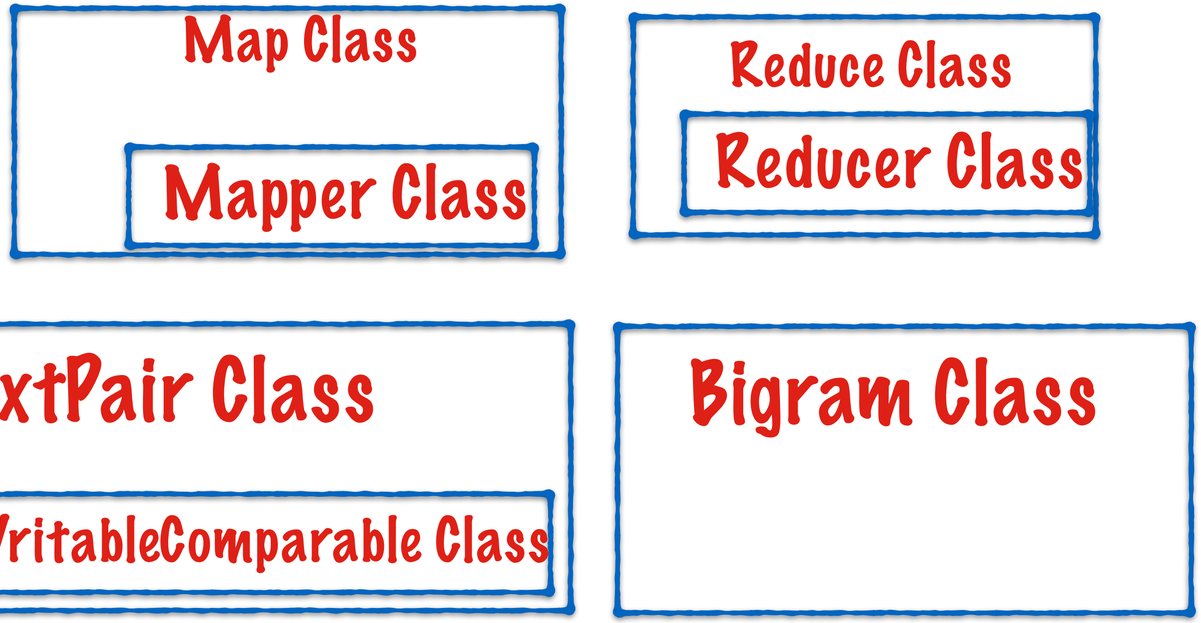
Reduce Class

Reducer Class

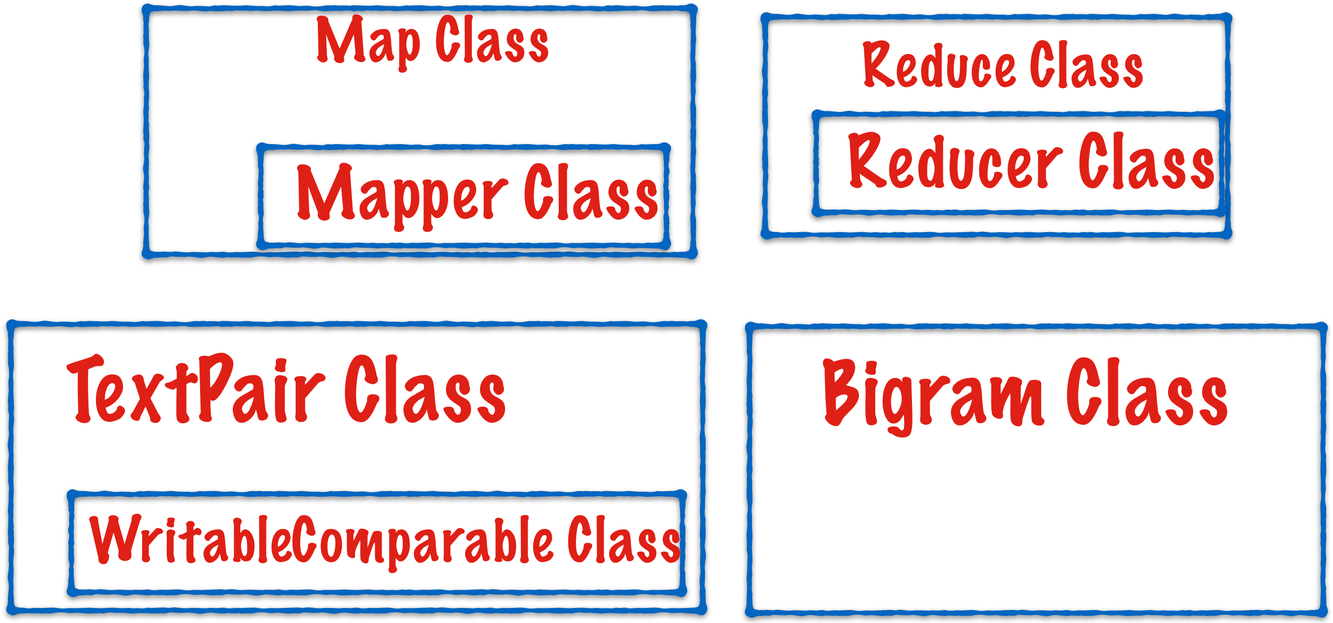
TextPair Class

WritableComparable Class

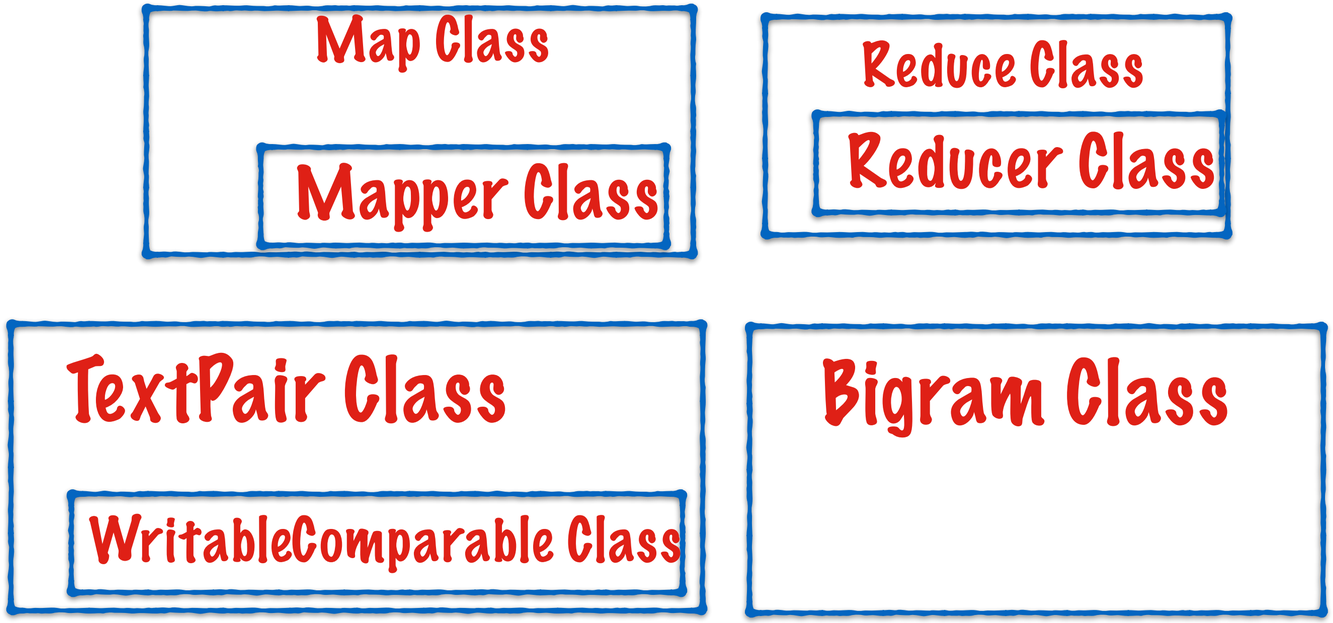
Bigram Class



We've written the code to compute Bigram Count



Let's write a Unit Test to test this code



Let's write a Unit Test to test this code

Unit Tests with MRUnit

Unit Tests with MRUnit

MRUnit is a testing framework for MapReduce

provided by APACHE

built using JUnit

Unit Tests with MRUnit

Download the MRUnit JAR from APACHE and add it to your project

https://repository.apache.org/content/repositories/releases/org/apache/mrunit/mrunit/

MRUnit is available for both Hadoop 1 and Hadoop 2

Unit Tests with MRUnit

Download the MRUnit JAR from APACHE and add it to your project

https://repository.apache.org/content/repositories/releases/org/apache/mrunit/mrunit/

We'll download the latest MRUnit JAR for Hadoop 2

Unit Tests with MRUnit

TestBigramCount Class

```
public class TestBigramCount {
    MapDriver<LongWritable, Text, TextPair, IntWritable> mapDriver;
    ReduceDriver<TextPair, IntWritable, Text, IntWritable> reduceDriver;
    MapReduceDriver<LongWritable, Text, TextPair, IntWritable, Text, IntWritable> mapReduceDriver;

    @Before
    public void setUp()
    {
        Map mapper = new Map();
        mapDriver = new MapDriver<LongWritable, Text, TextPair, IntWritable>();
        mapDriver.setMapper(mapper);

        Reduce reducer = new Reduce();
        reduceDriver = new ReduceDriver<TextPair, IntWritable, Text, IntWritable>();
        reduceDriver.setReducer(reducer);

        mapReduceDriver = new MapReduceDriver<LongWritable, Text, TextPair, IntWritable, Text,
IntWritable>();
        mapReduceDriver.setMapper(mapper);
        mapReduceDriver.setReducer(reducer);
    }

    @Test
    public void testMapper() throws IOException {
        mapDriver.withInput(new LongWritable(1), new Text("this is the bigram this is"));
        mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
        mapDriver.withOutput(new TextPair("is", "the"), new IntWritable(1));
        mapDriver.withOutput(new TextPair("the", "bigram"), new IntWritable(1));
        mapDriver.withOutput(new TextPair("bigram", "this"), new IntWritable(1));
        mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
        mapDriver.runTest();
    }

    @Test
    public void testReducer() throws IOException
    {
        List values = new ArrayList();
        values.add(new IntWritable(1));
        values.add(new IntWritable(1));
        reduceDriver.withInput(new TextPair("this","is"), values);
        reduceDriver.withOutput(new Text("this" +"\t" +"is"), new IntWritable(2));
```

Unit Tests with MRUnit

TestBigramCount Class

```
public class TestBigramCount {
MapDriver<LongWritable, Text, TextPair, IntWritable> mapDriver;
ReduceDriver<TextPair, IntWritable, Text, IntWritable> reduceDriver;
MapReduceDriver<LongWritable, Text, TextPair, IntWritable, Text, IntWritable>
mapReduceDriver;
```

```
@Before
public void setUp()
{
    Map mapper = new Map();
    mapDriver = new MapDriver<LongWritable, Text, TextPair, IntWritable>();
    mapDriver.setMapper(mapper);

    Reduce reducer = new Reduce();
    reduceDriver = new ReduceDriver<TextPair, IntWritable, Text, IntWritable>();
    reduceDriver.setReducer(reducer);

    mapReduceDriver = new MapReduceDriver<LongWritable, Text, TextPair, IntWritable, Text, IntWritable>();
    mapReduceDriver.setMapper(mapper);
    mapReduceDriver.setReducer(reducer);
}

@Test
public void testMapper() throws IOException {
    mapDriver.withInput(new LongWritable(1), new Text("this is the bigram this is"));
    mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("is", "the"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("the", "bigram"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("bigram", "this"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
    mapDriver.runTest();
}

@Test
public void testReducer() throws IOException
{
    List values = new ArrayList();
    values.add(new IntWritable(1));
    values.add(new IntWritable(1));
    reduceDriver.withInput(new TextPair("this", "is"), values);
```

MRUnit provides 3 drivers for testing MR jobs

Unit Tests with MRUnit

TestBigramCount Class

```
public class TestBigramCount {
MapDriver<LongWritable, Text, TextPair, IntWritable> mapDriver;
ReduceDriver<TextPair, IntWritable, Text, IntWritable> reduceDriver;
MapReduceDriver<LongWritable, Text, TextPair, IntWritable, Text, IntWritable>
mapReduceDriver;
```

```
@Before
public void setUp()
{
    Map mapper = new Map();
    mapDriver = new MapDriver<LongWritable, Text, TextPair, IntWritable>();
    mapDriver.setMapper(mapper);

    Reduce reducer = new Reduce();
    reduceDriver = new ReduceDriver<TextPair, IntWritable, Text, IntWritable>();
    reduceDriver.setReducer(reducer);

    mapReduceDriver = new MapReduceDriver<LongWritable, Text, TextPair, IntWritable, Text, IntWritable>();
    mapReduceDriver.setMapper(mapper);
    mapReduceDriver.setReducer(reducer);
}

@Test
public void testMapper() throws IOException {
    mapDriver.withInput(new LongWritable(1), new Text("this is the bigram this is"));
    mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("is", "the"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("the", "bigram"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("bigram", "this"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
    mapDriver.runTest();
}

@Test
public void testReducer() throws IOException
{
    List values = new ArrayList();
    values.add(new IntWritable(1));
    values.add(new IntWritable(1));
    reduceDriver.withInput(new TextPair("this", "is"), values);
```

MapDriver
ReduceDriver
MapReduceDriver

Unit Tests with MRUnit

TestBigramCount Class

```
public class TestBigramCount {
MapDriver<LongWritable, Text, TextPair, IntWritable> mapDriver;
ReduceDriver<TextPair, IntWritable, Text, IntWritable> reduceDriver;
MapReduceDriver<LongWritable, Text, TextPair, IntWritable, Text, IntWritable>
mapReduceDriver;
```

```
@Before
public void setUp()
{
    Map mapper = new Ma();
    mapDriver = new Map
    mapDriver.setMapper
    Reduce reducer = new Reduce();
    reduceDriver = new ReduceDriver<TextPair, IntWritable, Text, IntWritable>();
    reduceDriver.se ucer(reducer);
    mapReduceDriver = new
    mapReduceDriver.setMa
    mapReduceDriver.setReducer(reducer);
}

@Test
public vo
    mapDr
    mapDr
    mapDriver.withOutput(new TextPair("is", "the"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("the", "bigram"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("bigram", "this"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
    mapDriver.runTest();
}

@Test
public void testReducer() throws IOException
{
    List values = new ArrayList();
    values.add(new IntWritable(1));
    values.add(new IntWritable(1));
    reduceDriver.withInput(new TextPair("this", "is"), values);
```

MapDriver

ReduceDriver

MapReduceDriver

These are generics, whose input/output type parameters need to be specified

Unit Tests with MRUnit

TestBigramCount Class

```
public class TestBigramCount {
MapDriver<LongWritable, Text, TextPair, IntWritable> mapDriver;
ReduceDriver<TextPair, IntWritable, Text, IntWritable> reduceDriver;
MapReduceDriver<LongWritable, Text, TextPair, IntWritable, Text, IntWritable>
mapReduceDriver;
```

```
@Before
public void setUp()
{
    Map mapper = new Ma();
    mapDriver = new Map<LongWritable, Text, TextPair, IntWritable>();
    mapDriver.setMapper(mapper);

    Reduce reducer = new Reduce();
    reduceDriver = new ReduceDriver<TextPair, IntWritable, Text, IntWritable>();
    reduceDriver.setReducer(reducer);

    mapReduceDriver = new MapReduceDriver<LongWritable, Text, TextPair, IntWritable, Text, IntWritable>();
    mapReduceDriver.setMapper(mapper);
    mapReduceDriver.setReducer(reducer);
}

@Test
public void testMapper() throws IOException
{
    mapDriver.withInput(new LongWritable(1), new Text("this is the bigram test"));
    mapDriver.withOutput(new TextPair("is", "the"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("the", "bigram"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("bigram", "this"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
    mapDriver.runTest();
}

@Test
public void testReducer() throws IOException
{
    List values = new ArrayList();
    values.add(new IntWritable(1));
    values.add(new IntWritable(1));
    reduceDriver.withInput(new TextPair("this", "is"), values);
```

MapDriver

ReduceDriver

MapReduceDriver

The type parameters should match the types of the corresponding class

Unit Tests with MRUnit

TestBigramCount Class

```
public class TestBigramCount {
    MapDriver<LongWritable, Text, TextPair, IntWritable> mapDriver;
    ReduceDriver<TextPair, IntWritable, Text, IntWritable> reduceDriver;
    MapReduceDriver<LongWritable, Text, TextPair, IntWritable, Text, IntWritable> mapReduceDriver;
```

```
@Before
public void setUp()
```

```
    {
    Map mapper = new Map();
    mapDriver = new MapDriver<LongWritable, Text, TextPair, IntWritable>();
    mapDriver.setMapper(mapper);

    Reduce reducer = new Reduce();
    reduceDriver = new ReduceDriver<TextPair, IntWritable, Text, IntWritable>();
    reduceDriver.setReducer(reducer);

    mapReduceDriver = new MapReduceDriver<LongWritable, Text, TextPair, IntWritable, Text, IntWritable>();
    mapReduceDriver.setMapper(mapper);
    mapReduceDriver.setReducer(reducer);
}
```

We need to do some setup before we can run our tests

```
@Test
public void testMapper() throws IOException {
    mapDriver.withInput(new LongWritable(1), new Text("this is the bigram this is"));
    mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("is", "the"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("the", "bigram"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("bigram", "this"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
    mapDriver.runTest();
}

@Test
public void testReducer() throws IOException
{
    List values = new ArrayList();
    values.add(new IntWritable(1));
    values.add(new IntWritable(1));
```

Unit Tests with MRUnit

TestBigramCount Class

```
public class TestBigramCount {
    MapDriver<LongWritable, Text, TextPair, IntWritable> mapDriver;
    ReduceDriver<TextPair, IntWritable, Text, IntWritable> reduceDriver;
    MapReduceDriver<LongWritable, Text, TextPair, IntWritable, Text, IntWritable> mapReduceDriver;

    @Before
    public void setUp()
    {
        Map mapper = new Map();
        mapDriver = new MapDriver<LongWritable, Text, TextPair, IntWr
        mapDriver.setMapper(mapper);

        Reduce reducer = new Reduce();
        reduceDriver = new ReduceDriver<TextPair, IntWritable, Text, IntWritable>();
        reduceDriver.setReducer(reducer);

        mapReduceDriver = new MapReduceDriver<LongWritable, Text, TextPair, IntWritable, Text, IntWritable>();
        mapReduceDriver.setMapper(mapper);
        mapReduceDriver.setReducer(reducer);
    }

    @Test
    public void testMapper() throws IOException {
        mapDriver.withInput(new LongWritable(1), new Text("this is the bigram this is"));
        mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
        mapDriver.withOutput(new TextPair("is", "the"), new IntWritable(1));
        mapDriver.withOutput(new TextPair("the", "bigram"), new IntWritable(1));
        mapDriver.withOutput(new TextPair("bigram", "this"), new IntWritable(1));
        mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
        mapDriver.runTest();
    }

    @Test
    public void testReducer() throws IOException
    {
        List values = new ArrayList();
        values.add(new IntWritable(1));
        values.add(new IntWritable(1));
        reduceDriver.withInput(new TextPair("this","is"), values);
        reduceDriver.withOutput(new Text("this" +"\t" +"is"), new IntWritable(2));
        reduceDriver.runTest();
```

We setup a MapDriver using our Mapper class

Unit Tests with MRUnit

TestBigramCount Class

```
public class TestBigramCount {
    MapDriver<LongWritable, Text, TextPair, IntWritable> mapDriver;
    ReduceDriver<TextPair, IntWritable, Text, IntWritable> reduceDriver;
    MapReduceDriver<LongWritable, Text, TextPair, IntWritable, Text, IntWritable> mapReduceDriver;

    @Before
    public void setUp()
    {
        Map mapper = new Map();
        mapDriver = new MapDriver<LongWritable, Text, TextPair, IntWritable>();
        mapDriver.setMapper(mapper);

      Reduce reducer = new Reduce();
      reduceDriver = new ReduceDriver<TextPair, IntWritable, Text, IntWritable>();
      reduceDriver.setReducer(reducer);


        mapReduceDriver = new MapReduceDriver<LongWritable, Text, TextPair, IntWritable, Text, IntWritable>();
        mapReduceDriver.setMapper(mapper);
        mapReduceDriver.setReducer(reducer);
    }

    @Test
    public void testMapper() throws IOException {
        mapDriver.withInput(new LongWritable(1), new Text("this is the bigram this is"));
        mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
        mapDriver.withOutput(new TextPair("is", "the"), new IntWritable(1));
        mapDriver.withOutput(new TextPair("the", "bigram"), new IntWritable(1));
        mapDriver.withOutput(new TextPair("bigram", "this"), new IntWritable(1));
        mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
        mapDriver.runTest();
    }

    @Test
    public void testReducer() throws IOException
    {
        List values = new ArrayList();
        values.add(new IntWritable(1));
        values.add(new IntWritable(1));
        reduceDriver.withInput(new TextPair("this","is"), values);
        reduceDriver.withOutput(new Text("this" +"\t" +"is"), new IntWritable(2));
        reduceDriver.runTest();
```

We setup a ReduceDriver using our Reducer class

Unit Tests with MRUnit

TestBigramCount Class

```
public class TestBigramCount {
    MapDriver<LongWritable, Text, TextPair, IntWritable> mapDriver;
    ReduceDriver<TextPair, IntWritable, Text, IntWritable> reduceDriver;
    MapReduceDriver<LongWritable, Text, TextPair, IntWritable, Text, IntWritable> mapReduceDriver;

    @Before
    public void setUp()
    {
        Map mapper = new Map();
        mapDriver = new MapDriver<LongWritable, Text, TextPair, IntWritable>();
        mapDriver.setMapper(mapper);

      Reduce reducer = new Reduce();
      reduceDriver = new ReduceDriver<TextPair, IntWritable, Text, IntWritable>();

      reduceDriver.setReducer(reducer);
```

```
mapReduceDriver = new MapReduceDriver<LongWritable, Text, TextPair, IntWritable>();
mapReduceDriver.setMapper(mapper);
mapReduceDriver.setReducer(reducer);
```

```
    }

    @Test
    public void testMapper() throws IOException {
        mapDriver.withInput(new LongWritable(1), new Text("this is the bigram this is"));
        mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
        mapDriver.withOutput(new TextPair("is", "the"), new IntWritable(1));
        mapDriver.withOutput(new TextPair("the", "bigram"), new IntWritable(1));
        mapDriver.withOutput(new TextPair("bigram", "this"), new IntWritable(1));
        mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
        mapDriver.runTest();
    }

    @Test
    public void testReducer() throws IOException
    {
        List values = new ArrayList();
        values.add(new IntWritable(1));
        values.add(new IntWritable(1));
        reduceDriver.withInput(new TextPair("this","is"), values);
```

Finally a MapReduceDriver, for which we need to specify both the mapper and reducer

Unit Tests with MRUnit

TestBigramCount Class

```
p mapper = new Map();
pDriver = new MapDriver<LongWritable, Text, TextPair, IntWritable>();
pDriver.setMapper(mapper);

ce reducer = new Reduce();
ceDriver = new ReduceDriver<TextPair, IntWritable, Text, IntWritable>();

ceDriver.setReducer(reducer);


ReduceDriver = new MapReduceDriver<LongWritable, Text, TextPair, IntWritable, Text, IntWritable>();
ReduceDriver.setMapper(mapper);
ReduceDriver.setReducer(reducer);
```

```
@Test
public void testMapper() throws IOException {
mapDriver.withInput(new LongWritable(1), new Text("this is the bigram this is"));
     mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
     mapDriver.withOutput(new TextPair("is", "the"), new IntWritable(1));
     mapDriver.withOutput(new TextPair("the", "bigram"), new IntWritable(1));
     mapDriver.withOutput(new TextPair("bigram", "this"), new IntWritable(1));
     mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
     mapDriver.runTest();
}
```

```
 void testReducer() throws IOException

st values = new ArrayList();
lues.add(new IntWritable(1));
lues.add(new IntWritable(1));
duceDriver.withInput(new TextPair("this","is"), values);
duceDriver.withOutput(new Text("this" +"\t" +"is"), new IntWritable(2));
duceDriver.runTest();
```

```
 void testMapperReducer() throws IOException

pReduceDriver.addInput(new LongWritable(1), new Text("this is the bigram this is"));
pReduceDriver.addOutput(new Text("bigram" + "\t" + "this"), new IntWritable(1));
pReduceDriver.addOutput(new Text("is" + "\t" + "the"), new IntWritable(1));
pReduceDriver.addOutput(new Text("the" +"\t" +"bigram"), new IntWritable(1));
pReduceDriver.addOutput(new Text("this" + "\t" + "is"), new IntWritable(2));
pReduceDriver.runTest();
```

Here is a test for the Mapper we have written

Unit Tests with MRUnit

TestBigramCount Class

```
ap mapper = new Map();
apDriver = new MapDriver<LongWritable, Text, TextPair, IntWritable>();
apDriver.setMapper(mapper);

uce reducer = new Reduce();
uceDriver = new ReduceDriver<TextPair, IntWritable, Text, IntWritable>();

uceDriver.setReducer(reducer);


pReduceDriver = new MapReduceDriver<LongWritable, Text, TextPair, IntWritable, Text, IntWritable>();
pReduceDriver.setMapper(mapper);
pReduceDriver.setReducer(reducer);
```

```
@Test
public void testMapper() throws IOException {
mapDriver.withInput(new LongWritable(1), new Text("this is the bigram this is"));
    mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("is", "the"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("the", "bigram"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("bigram", "this"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
    mapDriver.runTest();
}
```

```
c void testReducer() throws IOException

ist values = new ArrayList();
alues.add(new IntWritable(1));
alues.add(new IntWritable(1));
educeDriver.withInput(new TextPair("this","is"), values);
educeDriver.withOutput(new Text("this" +"\t" +"is"), new IntWritable(2));
educeDriver.runTest();


c void testMapperReducer() throws IOException

apReduceDriver.addInput(new LongWritable(1), new Text("this is the bigram this is"));
apReduceDriver.addOutput(new Text("bigram" + "\t" + "this"), new IntWritable(1));
apReduceDriver.addOutput(new Text("is" + "\t" + "the"), new IntWritable(1));
apReduceDriver.addOutput(new Text("the" +"\t" +"bigram"), new IntWritable(1));
apReduceDriver.addOutput(new Text("this" + "\t" + "is"), new IntWritable(2));
apReduceDriver.runTest();
```

Precede the test with @Test, this is an annotation from JUnit

Unit Tests with MRUnit

TestBigramCount Class

```
ic void setUp()

Map mapper = new Map();
mapDriver = new MapDriver<LongWritable, Text, TextPair, IntWritable>();
mapDriver.setMapper(mapper);

duce reducer = new Reduce();
duceDriver = new ReduceDriver<TextPair, IntWritable, Text, IntWritable>();
duceDriver.setReducer(reducer);

pReduceDriver = new MapReduceDriver<LongWritable, Text, TextPair, IntWritable, Text, IntWritable>();
pReduceDriver.setMapper(mapper);
pReduceDriver.setReducer(reducer);
```

```
@Test
public void testMapper() throws IOException {
mapDriver.withInput(new LongWritable(1), new Text("this is the bigram this is"));
    mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("is", "the"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("the", "bigram"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("bigram", "this"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
    mapDriver.runTest();
}
```

```
t
ic void testReducer() throws IOException

List values = new ArrayList();
values.add(new IntWritable(1));
values.add(new IntWritable(1));
reduceDriver.withInput(new TextPair("this","is"), values);
reduceDriver.withOutput(new Text("this" +"\t" +"is"), new IntWritable(2));
reduceDriver.runTest();
```

```
t
ic void testMapperReducer() throws IOException

mapReduceDriver.addInput(new LongWritable(1), new Text("this is the bigram this is"));
mapReduceDriver.addOutput(new Text("bigram" + "\t" + "this"), new IntWritable(1));
mapReduceDriver.addOutput(new Text("is" + "\t" + "the"), new IntWritable(1));
mapReduceDriver.addOutput(new Text("the" +"\t" +"bigram"), new IntWritable(1));
mapReduceDriver.addOutput(new Text("this" + "\t" + "is"), new IntWritable(2));
mapReduceDriver.runTest();
```

We specify and input

Unit Tests with MRUnit

TestBigramCount Class

```
ic void setUp()

Map mapper = new Map();
mapDriver = new MapDriver<LongWritable, Text, TextPair, IntWritable>();
mapDriver.setMapper(mapper);

duce reducer = new Reduce();
duceDriver = new ReduceDriver<TextPair, IntWritable, Text, IntWritable>();
duceDriver.setReducer(reducer);

pReduceDriver = new MapReduceDriver<LongWritable, Text, TextPair, IntWritable, Text, IntWritable>();
pReduceDriver.setMapper(mapper);
pReduceDriver.setReducer(reducer);
```

```
@Test
public void testMapper() throws IOException {
mapDriver.withInput(new LongWritable(1), new Text("this is the bigram this is"));
    mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("is", "the"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("the", "bigram"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("bigram", "this"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
    mapDriver.runTest();
}
```

```
t
ic void testReducer() throws IOException

List values = new ArrayList();
values.add(new IntWritable(1));
values.add(new IntWritable(1));
reduceDriver.withInput(new TextPair("this","is"), values);
reduceDriver.withOutput(new Text("this" +"\t" +"is"), new IntWritable(2));
reduceDriver.runTest();
```

```
t
ic void testMapperReducer() throws IOException

mapReduceDriver.addInput(new LongWritable(1), new Text("this is the bigram this is"));
mapReduceDriver.addOutput(new Text("bigram" + "\t" + "this"), new IntWritable(1));
mapReduceDriver.addOutput(new Text("is" + "\t" + "the"), new IntWritable(1));
mapReduceDriver.addOutput(new Text("the" +"\t" +"bigram"), new IntWritable(1));
mapReduceDriver.addOutput(new Text("this" + "\t" + "is"), new IntWritable(2));
mapReduceDriver.runTest();
```

In the output consists of all the bigrams for this sentence as keys

Unit Tests with MRUnit

TestBigramCount Class

```
@Test
public void testMapper() throws IOException {

mapDriver.withInput(new LongWritable(1), new Text("this is the bigram this is"));
    mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("is", "the"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("the", "bigram"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("bigram", "this"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
    mapDriver.runTest();
}
```

The order of the output matters

Unit Tests with MRUnit

TestBigramCount Class

```
@Test
public void testMapper() throws IOException {

mapDriver.withInput(new LongWritable(1), new Text("this is the bigram this is"));
    mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("is", "the"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("the", "bigram"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("bigram", "this"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
    mapDriver.runTest();
}
```

The order of the output matters

Unit Tests with MRUnit

TestBigramCount Class

```
@Test
public void testMapper() throws IOException {

mapDriver.withInput(new LongWritable(1), new Text("this is the bigram this i
    mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("is", "the"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("the", "bigram"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("bigram", "this"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
    mapDriver.runTest();
}
```

.. and so on

Unit Tests with MRUnit

TestBigramCount Class

```
MapReduceDriver<LongWritable, Text, TextPair, IntWritable, Text, IntWritable> mapReduceDriver;

@Before
public void setUp()
{
    Map mapper = new Map();
    mapDriver = new MapDriver<LongWritable, Text, TextPair, IntWritable>();
    mapDriver.setMapper(mapper);
    Reducer reducer = new Reducer();
    reduceDriver = new ReduceDriver<TextPair, IntWritable, Text, IntWritable>();
    reduceDriver.setReducer(reducer);

mapReduceDriver = new MapReduceDriver<LongWritable, Text, TextPair, IntWritable, Text, IntWritable>();
    mapReduceDriver.setMapper(mapper);
    mapReduceDriver.setReducer(reducer);
}

@Test
public void testMapper() throws IOException {
mapDriver.withInput(new LongWritable(1), new Text("this is the bigram this is"));
    mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("is", "the"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("the", "bigram"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("bigram", "this"), new IntWritable(1));

    mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));

        mapDriver.runTest();
();

}

@Test
public void testReducer() throws IOException
{
    List values = new ArrayList();
    values.add(new IntWritable(1));
    values.add(new IntWritable(1));
    reduceDriver.withInput(new TextPair("this","is"), values);
    reduceDriver.withOutput(new Text("this" +"\t" +"is"), new IntWritable(2));
    reduceDriver.runTest();
}

@Test
public void testMapperReducer() throws IOException
{
    mapReduceDriver.addInput(new LongWritable(1), new Text("this is the bigram this is"));
    mapReduceDriver.addOutput(new Text("bigram" + "\t" + "this"), new IntWritable(1));
    mapReduceDriver.addOutput(new Text("is" + "\t" + "the"), new IntWritable(1));
    mapReduceDriver.addOutput(new Text("the" +"\t" +"bigram"), new IntWritable(1));
    mapReduceDriver.addOutput(new Text("this" + "\t" + "is"), new IntWritable(2));
    mapReduceDriver.runTest();
}
```

Finally we can run the test

```
public void testMapper() throws IOException {
    mapDriver.withInput(new LongWritable(1), new Text("this is the bigram this is"));
    mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("is", "the"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("the", "bigram"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("bigram", "this"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
    mapDriver.runTest();
```

Unit Tests with MRUnit

TestBigramCount Class

```
@Test
public void testReducer() throws IOException
{
    List values = new ArrayList();
    values.add(new IntWritable(1));
    values.add(new IntWritable(1));
    reduceDriver.withInput(new TextPair("this","is"), values);
    reduceDriver.withOutput(new Text("this" +"\t" +"is"), new IntWritable(2));
    reduceDriver.runTest();
}
```

```
@Test
public void testMapperReducer() throws IOException
{
    mapReduceDriver.addInput(new LongWritable(1), new Text("this is the bigram this is"));
    mapReduceDriver.addOutput(new Text("bigram" + "\t" + "this"), new IntWritable(1));
    mapReduceDriver.addOutput(new Text("is" + "\t" + "the"), new IntWritable(1));
    mapReduceDriver.addOutput(new Text("the" +"\t" +"bigram"), new IntWritable(1));
    mapReduceDriver.addOutput(new Text("this" + "\t" + "is"), new IntWritable(2));
    mapReduceDriver.runTest();
}
```

Similarly we can write a test for the reducer

```
public void testMapper() throws IOException {
    mapDriver.withInput(new LongWritable(1), new Text("this is the bigram this is"));
    mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("is", "the"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("the", "bigram"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("bigram", "this"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
    mapDriver.runTest();
```

Unit Tests with MRUnit

TestBigramCount Class

```
@Test
public void testReducer() throws IOException
{
    List values = new ArrayList();
    values.add(new IntWritable(1));
    values.add(new IntWritable(1));
    reduceDriver.withInput(new TextPair("this","is"), values);
    reduceDriver.withOutput(new Text("this" +"\t" +"is"), new IntWritable(2));
    reduceDriver.runTest();
}
```

```
@Test
public void testMapperReducer() throws IOException
{
    mapReduceDriver.addInput(new LongWritable(1), new Text("this is the bigram this is"));
    mapReduceDriver.addOutput(new Text("bigram" + "\t" + "this"), new IntWritable(1));
    mapReduceDriver.addOutput(new Text("is" + "\t" + "the"), new IntWritable(1));
    mapReduceDriver.addOutput(new Text("the" +"\t" +"bigram"), new IntWritable(1));
    mapReduceDriver.addOutput(new Text("this" + "\t" + "is"), new IntWritable(2));
    mapReduceDriver.runTest();
}
```

The reducer needs a TextPair as input key

```
public void testMapper() throws IOException {
    mapDriver.withInput(new LongWritable(1), new Text("this is the bigram this is"));
    mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("is", "the"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("the", "bigram"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("bigram", "this"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
    mapDriver.runTest();
```

Unit Tests with MRUnit

TestBigramCount Class

```
@Test
public void testReducer() throws IOException
{
    List values = new ArrayList();
    values.add(new IntWritable(1));
    values.add(new IntWritable(1));
    reduceDriver.withInput(new TextPair("this","is"), values);
    reduceDriver.withOutput(new Text("this" +"\t" +"is"), new IntWritable(2));
    reduceDriver.runTest();
}
```

```
@Test
public void testMapperReducer() throws IOException
{
    mapReduceDriver.addInput(new LongWritable(1), new Text("this is the bigram this is"));
    mapReduceDriver.addOutput(new Text("bigram" + "\t" + "this"), new IntWritable(1));
    mapReduceDriver.addOutput(new Text("is" + "\t" + "the"), new IntWritable(1));
    mapReduceDriver.addOutput(new Text("the" +"\t" +"bigram"), new IntWritable(1));
    mapReduceDriver.addOutput(new Text("this" + "\t" + "is"), new IntWritable(2));
    mapReduceDriver.runTest();
}
```

And a list for the input value

```
public void testMapper() throws IOException {
    mapDriver.withInput(new LongWritable(1), new Text("this is the bigram this is"));
    mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("is", "the"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("the", "bigram"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("bigram", "this"), new IntWritable(1));
    mapDriver.withOutput(new TextPair("this", "is"), new IntWritable(1));
    mapDriver.runTest();
```

Unit Tests with MRUnit

TestBigramCount Class

```
@Test
public void testReducer() throws IOException
{
    List values = new ArrayList();
    values.add(new IntWritable(1));
    values.add(new IntWritable(1));
    reduceDriver.withInput(new TextPair("this", "is"), values);
    reduceDriver.withOutput(new Text("this" +"\t" +"is"), new IntWritable(2));
    reduceDriver.runTest();
}
```

```
@Test
public void testMapperReducer() throws IOException
{
    mapReduceDriver.addInput(new LongWritable(1), new Text("this is the bigram this is"));
    mapReduceDriver.addOutput(new Text("bigram" + "\t" + "this"), new IntWritable(1));
    mapReduceDriver.addOutput(new Text("is" + "\t" + "the"), new IntWritable(1));
    mapReduceDriver.addOutput(new Text("the" +"\t" +"bigram"), new IntWritable(1));
    mapReduceDriver.addOutput(new Text("this" + "\t" + "is"), new IntWritable(2));
    mapReduceDriver.runTest();
}
```

The output key has to be of type Text and the output value is the count

Unit Tests with MRUnit

TestBigramCount Class

```
    @Test
    public void testReducer() throws IOException
    {
        List values = new ArrayList();
        values.add(new IntWritable(1));
        values.add(new IntWritable(1));
        reduceDriver.withInput(new TextPair("this","is"), values);
        reduceDriver.withOutput(new Text("this\tis"), new IntWritable(2));
        reduceDriver.runTest();
    }
```

```
    @Test
    public void testMapperReducer() throws IOException
    {
        mapReduceDriver.addInput(new LongWritable(1), new Text("this is the bigram this
is"));
        mapReduceDriver.addOutput(new Text("bigram" + "\t" + "this"), new IntWritable(1));
        mapReduceDriver.addOutput(new Text("is" + "\t" + "the"), new IntWritable(1));
        mapReduceDriver.addOutput(new Text("the" +"\t" +"bigram"), new IntWritable(1));
        mapReduceDriver.addOutput(new Text("this" + "\t" + "is"), new IntWritable(2));
        mapReduceDriver.runTest();
    }

}
```

Finally we have a test for the complete MapReduce job

Unit Tests with MRUnit

TestBigramCount Class

```
@Test
public void testReducer() throws IOException
{
    List values = new ArrayList();
    values.add(new IntWritable(1));
    values.add(new IntWritable(1));
    reduceDriver.withInput(new TextPair("this","is"), values);
    reduceDriver.withOutput(new TextPair("this","is"), new IntWritable(2));
    reduceDriver.runTest();
}
```

```
@Test
public void testMapperReducer() throws IOException
{
    mapReduceDriver.addInput(new LongWritable(1), new Text("this is the bigram this is"));
    mapReduceDriver.addOutput(new Text("bigram" + "\t" + "this"), new IntWritable(1));
    mapReduceDriver.addOutput(new Text("is" + "\t" + "the"), new IntWritable(1));
    mapReduceDriver.addOutput(new Text("the" +"\t" +"bigram"), new IntWritable(1));
    mapReduceDriver.addOutput(new Text("this" + "\t" + "is"), new IntWritable(2));
    mapReduceDriver.runTest();
}
}
```

The input is a sentence

Unit Tests with MRUnit

TestBigramCount Class

```
    @Test
    public void testMapperReducer() throws IOException
    {
        mapReduceDriver.addInput(new LongWritable(1), new Text("this is the bigram this
is"));
        mapReduceDriver.addOutput(new Text("bigram" + "\t" + "this"), new IntWritable(1));
        mapReduceDriver.addOutput(new Text("is" + "\t" + "the"), new IntWritable(1));
        mapReduceDriver.addOutput(new Text("the" +"\t" +"bigram"), new IntWritable(1));
        mapReduceDriver.addOutput(new Text("this" + "\t" + "is"), new IntWritable(2));
        mapReduceDriver.runTest();
    }

}
```

The output has the bigram counts for all bigrams in the sentence

Unit Tests with MRUnit

TestBigramCount Class

```
@Test
public void testReducer() throws IOException
{
    List values = new ArrayList();
    values.add(new IntWritable(1));
    values.add(new IntWritable(1));
    reduceDriver.withInput(new TextPair("this","is"), values);
    reduceDriver.withOutput(new TextPair("this","\t","is"), new IntWritable(2));
    reduceDriver.runTest();
}
```

```
@Test
public void testMapperReducer() throws IOException
{
    mapReduceDriver.addInput(new LongWritable(1), new Text("this is the bigram this
is"));
    mapReduceDriver.addOutput(new Text("bigram" + "\t" + "this"), new IntWritable(1));
    mapReduceDriver.addOutput(new Text("is" + "\t" + "the"), new IntWritable(1));
    mapReduceDriver.addOutput(new Text("the" +"\t" +"bigram"), new IntWritable(1));
    mapReduceDriver.addOutput(new Text("this" + "\t" + "is"), new IntWritable(2));
    mapReduceDriver.runTest();
}

}
```

The order of the output matters

Unit Tests with MRUnit

TestBigramCount Class

```
    @Test
    public void testReducer() throws IOException
    {
        List values = new ArrayList();
        values.add(new IntWritable(1));
        values.add(new IntWritable(1));
        reduceDriver.withInput(new TextPair("this","is"), values);
        reduceDriver.withOutput(new Text("this\tis"), new IntWritable(2));
        reduceDriver.runTest();
    }
```

```
    @Test
    public void testMapperReducer() throws IOException
    {
        mapReduceDriver.addInput(new LongWritable(1), new Text("this is the bigram this
is"));
        mapReduceDriver.addOutput(new Text("bigram" + "\t" + "this"), new IntWritable(1));
        mapReduceDriver.addOutput(new Text("is" + "\t" + "the"), new IntWritable(1));
        mapReduceDriver.addOutput(new Text("the" +"\t" +"bigram"), new IntWritable(1));
        mapReduceDriver.addOutput(new Text("this" + "\t" + "is"), new IntWritable(2));
        mapReduceDriver.runTest();
    }

}
```

It has to be in the sort order that Hadoop will sort in

Unit Tests with MRUnit

TestBigramCount Class

```
@Test
public void testReducer() throws IOException
{
    List values = new ArrayList();
    values.add(new IntWritable(1));
    values.add(new IntWritable(1));
    reduceDriver.withInput(new TextPair("this","is"), values);
    reduceDriver.withOutput(new TextPair("this","\t","is"), new IntWritable(2));
    reduceDriver.runTest();
}
```

```
@Test
public void testMapperReducer() throws IOException
{
    mapReduceDriver.addInput(new LongWritable(1), new Text("this is the bigram this
is"));
    mapReduceDriver.addOutput(new Text("bigram" + "\t" + "this"), new IntWritable(1));
    mapReduceDriver.addOutput(new Text("is" + "\t" + "the"), new IntWritable(1));
    mapReduceDriver.addOutput(new Text("the" +"\t" +"bigram"), new IntWritable(1));
    mapReduceDriver.addOutput(new Text("this" + "\t" + "is"), new IntWritable(2));
    mapReduceDriver.runTest();
}

}
```

We wrote a compareTo() method that sorts the bigrams by the first word, then second word

Unit Tests with MRUnit

TestBigramCount Class

```
    @Test
    public void testMapperReducer() throws IOException
    {
        mapReduceDriver.addInput(new LongWritable(1), new Text("this is the bigram this
is"));
        mapReduceDriver.addOutput(new Text("bigram" + "\t" + "this"), new IntWritable(1));
        mapReduceDriver.addOutput(new Text("is" + "\t" + "the"), new IntWritable(1));
        mapReduceDriver.addOutput(new Text("the" +"\t" +"bigram"), new IntWritable(1));
        mapReduceDriver.addOutput(new Text("this" + "\t" + "is"), new IntWritable(2));
        mapReduceDriver.runTest();
    }

}
```

So the output should follow that order

Unit Tests with MRUnit

Once you've written the Unit Tests,
you can use your IDE to run the tests

Your IDE needs to be configured to run JUnit tests

Unit Tests with MRUnit

The IntelliJ IDE is configured to run these tests by default

On an IDE like Eclipse, you should download and add the JUnit JAR and in the Run options, you'll see an option for running Unit tests