1. Write a MapReduce program to count the number of occurences of each word and provide output as follows:
Output
Word Word Count
a 1 (As the word ‘a’ occurred only once)
this 2 (As the word ‘this’ occurred twice)

```
package cdac_hadoop;

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

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

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

	public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException{
	StringTokenizer itr = new StringTokenizer(value.toString());

	while (itr.hasMoreTokens()){
		word.set(itr.nextToken());
		context.write(word, one);
	}
	}
}

package cdac_hadoop;

import java.io.IOException;

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

public class WCreducer extends Reducer<Text, IntWritable, Text, IntWritable>{
	private IntWritable result = new IntWritable();

	public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException
	{
		int sum = 0;
		for (IntWritable val: values){
			sum = sum + val.get();
		}
```

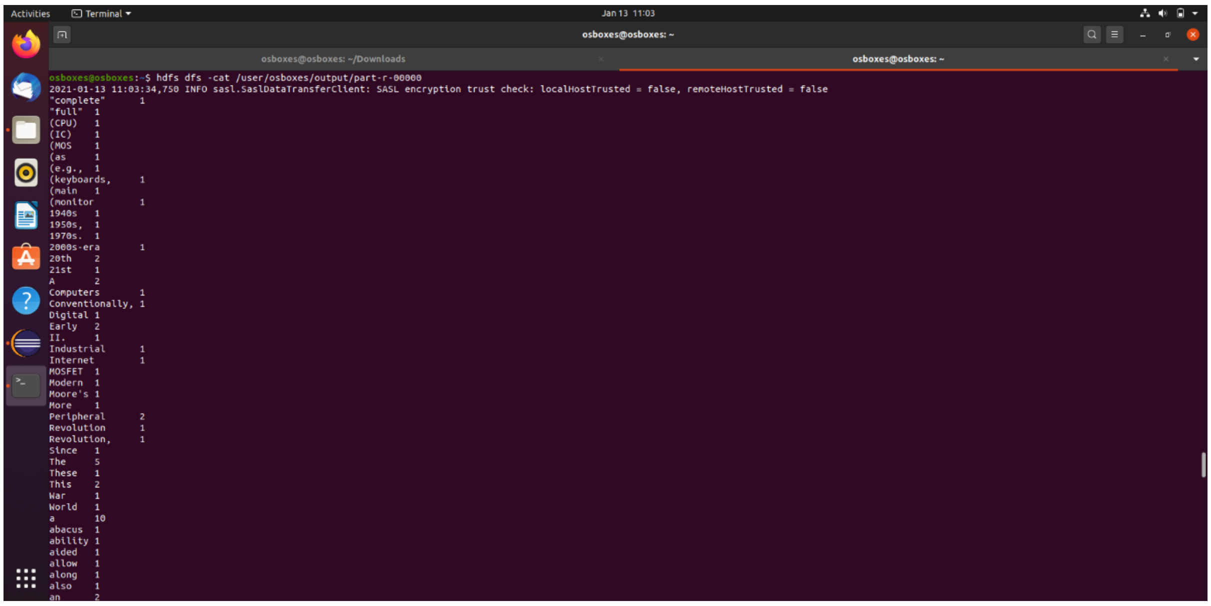
```
			result.set(sum);
			context.write(key, result);
		}

}

package cdac_hadoop;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
public class WordCount {

		public static void main(String[] args) throws Exception {
			Configuration conf = new Configuration();
			Job job = Job.getInstance(conf, "word Count");

			job.setJarByClass(WordCount.class);
			job.setMapperClass(WCmapper.class);
			job.setReducerClass(WCreducer.class);
			job.setOutputKeyClass(Text.class);
			job.setOutputValueClass(IntWritable.class);

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

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

}
```

Export java code as jar file.

Put the file in hdfs system
Command :- hdfs dfs -put computer.txt /user/osboxes/input/

Then use the jar file to run perform mapreduce program on above loaded text file.
Hadoop jar WC.jar /user/osboxes/input/computer.txt /user/osboxes/output/

Output:-



```
osboxes@osboxes:~$ hdfs dfs -cat /user/osboxes/output/part-r-00000
2021-01-13 11:03:34,750 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
"complete"      1
"full"  1
(CPU)   1
(IC)    1
(MOS    1
(as     1
(e.g.,  1
(keyboards,     1
(main   1
(monitor        1
1940s   1
1950s,  1
1970s.  1
2000s-era       1
20th    2
21st    1
A       2
Computers       1
Conventionally, 1
Digital 1
Early   2
II.     1
Industrial      1
Internet        1
MOSFET  1
Modern  1
Moore's 1
More    1
Peripheral      2
Revolution      1
Revolution,     1
Since   1
The     5
These   1
This    2
War     1
World   1
a       10
abacus  1
ability 1
aided   1
allow   1
along   1
also    1
an      2
```

2. Write a MapReducep rogramt hat reads the alphabetst ext file and counts the occurences of words of each size. The output should appear as follows:
Sample Output
Word Size Word Count
1 1 (As the word of size 1 is: a)
2 4 (As the words of size 2 are: is, of, of, in)
3 3 (As the words of size 3 are: the, and, the)
4 6 (As the words of size 4 are: this, word, size, that, size)

```
package Wordlength;
import java.io.IOException;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
```

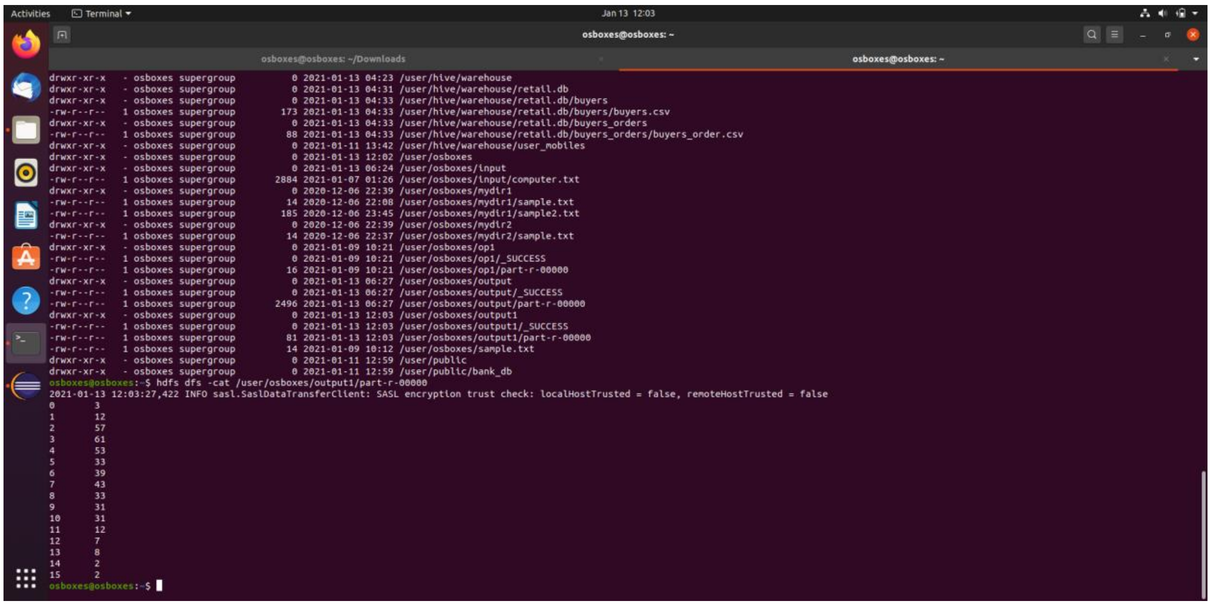
```
public class AlphabetWordCount {

public static class AlphabetWordCountMapper extends
Mapper<LongWritable,Text,IntWritable,IntWritable>{

 // here we are declaring a final static local variable of static type

 public static final IntWritable ONE = new IntWritable(1);

  @Override

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

   String line = value.toString();

   for (String word : line.split(" ")) {

       System.out.println("This is a Alphabet count problem");

    cont.write(new IntWritable(word.length()), ONE);

    }

  }

 }

 public static class AlphabetWordCountReducer extends Reducer<IntWritable, IntWritable,
IntWritable, IntWritable>{

  @Override

  public void reduce(IntWritable key, Iterable<IntWritable> value, Context cont) throws
IOException,InterruptedException{

   int count = 0;

   for (IntWritable values : value) {

   count += values.get();

     }

   cont.write(key, new IntWritable(count));

   }

   }
 public static void main(String[] args) throws
ClassNotFoundException,IOException,InterruptedException{

  // TODO Auto-generated method stub

  Configuration conf = new Configuration();

  Job job = Job.getInstance(conf, "WordCount");

  job.setMapperClass(AlphabetWordCountMapper.class);

  job.setReducerClass(AlphabetWordCountReducer.class);

  job.setCombinerClass(AlphabetWordCountReducer.class);

  job.setMapOutputKeyClass(IntWritable.class);

  job.setMapOutputValueClass(IntWritable.class);

  job.setOutputKeyClass(IntWritable.class);

  job.setOutputValueClass(IntWritable.class);

  job.setJarByClass(AlphabetWordCount.class);

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

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

  System.exit(job.waitForCompletion(true) ? 0 : 1);

 }
}
```

Export java code as jar file.

Put the file in hdfs system
Command :- hdfs dfs -put computer.txt /user/osboxes/input/

Then use the jar file to run perform mapreduce program on above loaded text file.
Hadoop jar Wordlengh.jar /user/osboxes/input/computer.txt /user/osboxes/output1/

Output:-



```
drwxr-xr-x   - osboxes supergroup          0 2021-01-13 04:23 /user/hive/warehouse
drwxr-xr-x   - osboxes supergroup          0 2021-01-13 04:31 /user/hive/warehouse/retail.db
drwxr-xr-x   - osboxes supergroup          0 2021-01-13 04:33 /user/hive/warehouse/retail.db/buyers
-rw-r--r--   1 osboxes supergroup        173 2021-01-13 04:33 /user/hive/warehouse/retail.db/buyers/buyers.csv
drwxr-xr-x   - osboxes supergroup          0 2021-01-13 04:33 /user/hive/warehouse/retail.db/buyers_orders
-rw-r--r--   1 osboxes supergroup         88 2021-01-13 04:33 /user/hive/warehouse/retail.db/buyers_orders/buyers_order.csv
drwxr-xr-x   - osboxes supergroup          0 2021-01-11 13:42 /user/hive/warehouse/user_mobiles
drwxr-xr-x   - osboxes supergroup          0 2021-01-13 12:02 /user/osboxes
drwxr-xr-x   - osboxes supergroup          0 2021-01-13 06:24 /user/osboxes/input
-rw-r--r--   1 osboxes supergroup       2884 2021-01-07 01:26 /user/osboxes/input/computer.txt
drwxr-xr-x   - osboxes supergroup          0 2020-12-06 22:39 /user/osboxes/mydir1
-rw-r--r--   1 osboxes supergroup         14 2020-12-06 22:08 /user/osboxes/mydir1/sample.txt
-rw-r--r--   1 osboxes supergroup        185 2020-12-06 23:45 /user/osboxes/mydir1/sample2.txt
drwxr-xr-x   - osboxes supergroup          0 2020-12-06 22:39 /user/osboxes/mydir2
-rw-r--r--   1 osboxes supergroup         14 2020-12-06 22:37 /user/osboxes/mydir2/sample.txt
drwxr-xr-x   - osboxes supergroup          0 2021-01-09 10:21 /user/osboxes/op1
-rw-r--r--   1 osboxes supergroup          0 2021-01-09 10:21 /user/osboxes/op1/_SUCCESS
-rw-r--r--   1 osboxes supergroup         16 2021-01-09 10:21 /user/osboxes/op1/part-r-00000
drwxr-xr-x   - osboxes supergroup          0 2021-01-13 06:27 /user/osboxes/output
-rw-r--r--   1 osboxes supergroup          0 2021-01-13 06:27 /user/osboxes/output/_SUCCESS
-rw-r--r--   1 osboxes supergroup       2496 2021-01-13 06:27 /user/osboxes/output/part-r-00000
drwxr-xr-x   - osboxes supergroup          0 2021-01-13 12:03 /user/osboxes/output1
-rw-r--r--   1 osboxes supergroup          0 2021-01-13 12:03 /user/osboxes/output1/_SUCCESS
-rw-r--r--   1 osboxes supergroup         81 2021-01-13 12:03 /user/osboxes/output1/part-r-00000
-rw-r--r--   1 osboxes supergroup         14 2021-01-09 10:12 /user/osboxes/sample.txt
drwxr-xr-x   - osboxes supergroup          0 2021-01-11 12:59 /user/public
drwxr-xr-x   - osboxes supergroup          0 2021-01-11 12:59 /user/public/bank_db
osboxes@osboxes:~$ hdfs dfs -cat /user/osboxes/output1/part-r-00000
2021-01-13 12:03:27,422 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
0       3
1       12
2       57
3       61
4       53
5       33
6       39
7       43
8       33
9       31
10      31
11      12
12      7
13      8
14      2
15      2
osboxes@osboxes:~$ 
```

3. Write a MapReduce program to process a given patent dataset with patent records. Each patent has sub-patent id's associated with it. You need to calculate the number of sub-patents associated with each patent.

```java
package patent;

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

	import org.apache.hadoop.io.IntWritable;
	import org.apache.hadoop.io.LongWritable;
	import org.apache.hadoop.io.Text;
	import org.apache.hadoop.mapreduce.Mapper;

	/**
	 * @author osboxes
	 *
	 */
	public class patentmapper extends Mapper<LongWritable,Text,Text,IntWritable> {
		private final static IntWritable one = new IntWritable(1);
		private Text key1 = new Text();
		private Text value1 = new Text();


		public void map(LongWritable key,Text value,Context context) throws
IOException, InterruptedException {
			StringTokenizer itr = new StringTokenizer(value.toString());

			while (itr.hasMoreTokens()) {
				String patent = itr.nextToken();
				key1.set(patent);
```

```
                        String subpatent = itr.nextToken();
                        value1.set(subpatent);
                        context.write(key1, one);
                    }
                }

            }

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
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;

public class pat {

    public static void main(String[] args) throws Exception  {
        // TODO Auto-generated method stub

        Configuration conf = new Configuration();

        Job job = Job.getInstance(conf,"patent");

        job.setJarByClass(pat.class);
        job.setMapperClass(patentmapper.class);
        job.setReducerClass(patred.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

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

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
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;

public class pat {

    public static void main(String[] args) throws Exception  {
        // TODO Auto-generated method stub
```

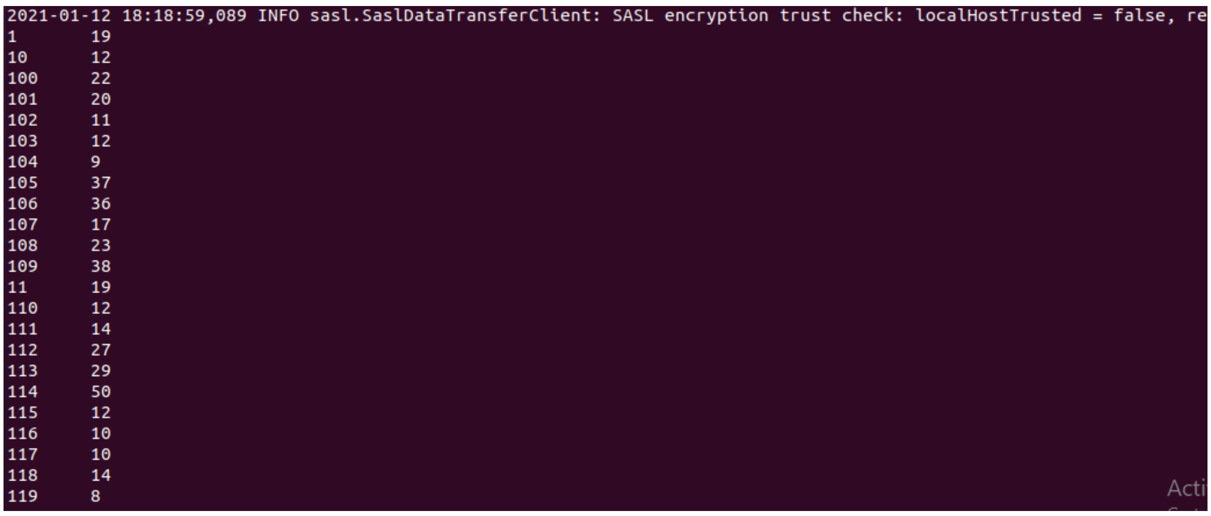
```
                    Configuration conf = new Configuration();

                    Job job = Job.getInstance(conf,"patent");

                    job.setJarByClass(pat.class);
                    job.setMapperClass(patentmapper.class);
                    job.setReducerClass(patred.class);
                    job.setOutputKeyClass(Text.class);
                    job.setOutputValueClass(IntWritable.class);

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

                    System.exit(job.waitForCompletion(true) ? 0 : 1);
          }
}
```

Export java code as jar file.

Put the file in hdfs system
Command :- hdfs dfs -put patent /user/osboxes/input/

Then use the jar file to run perform mapreduce program on above loaded text file.
Hadoop jar WC.jar /user/osboxes/input/computer.txt /user/osboxes/output2/

Output:-

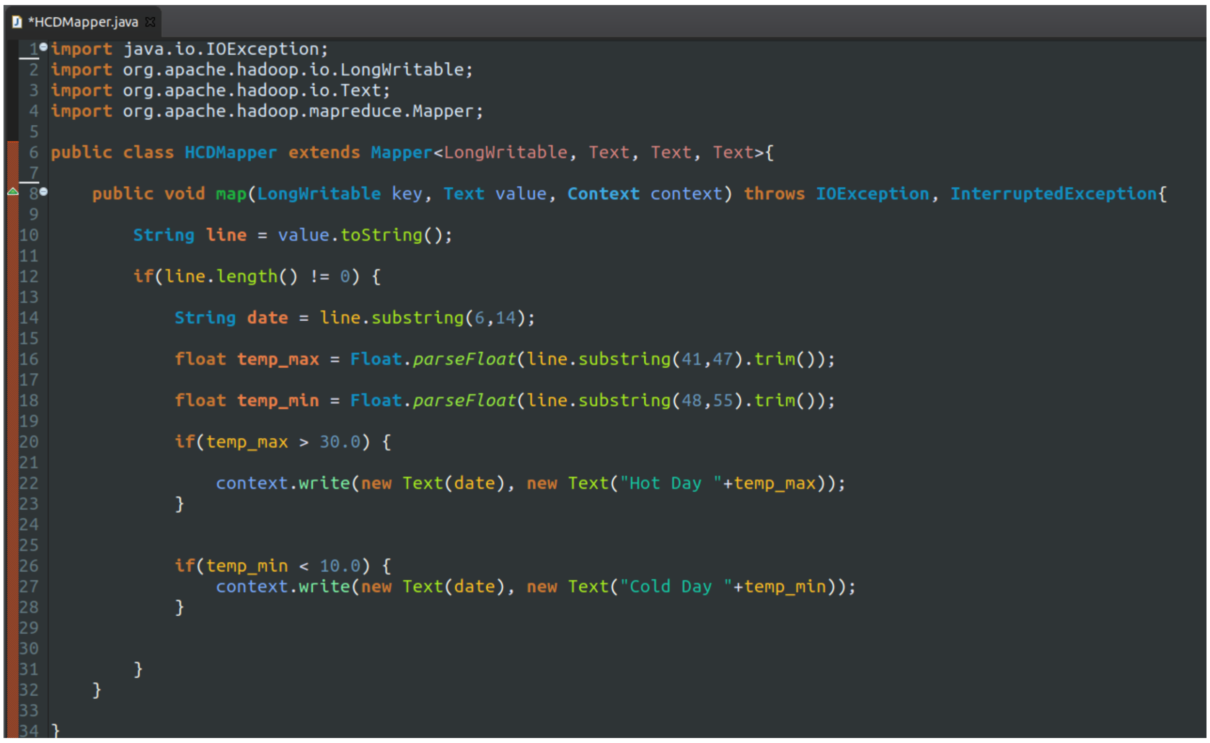


```
2021-01-12 18:18:59,089 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, re
1       19
10      12
100     22
101     20
102     11
103     12
104     9
105     37
106     36
107     17
108     23
109     38
11      19
110     12
111     14
112     27
113     29
114     50
115     12
116     10
117     10
118     14
119     8
```

4. Write a MapReduce program to find out the dates with maximum temperature greater than 40 (A Hot Day) and minimum temperature lower than 10 (A Cold Day).

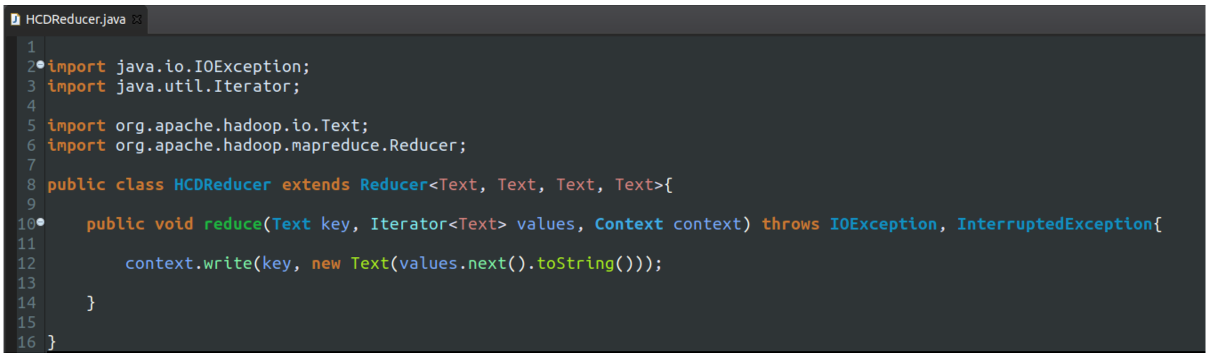
Take WeatherData.txt as input file.

Mapper class :


*HCDMapper.java

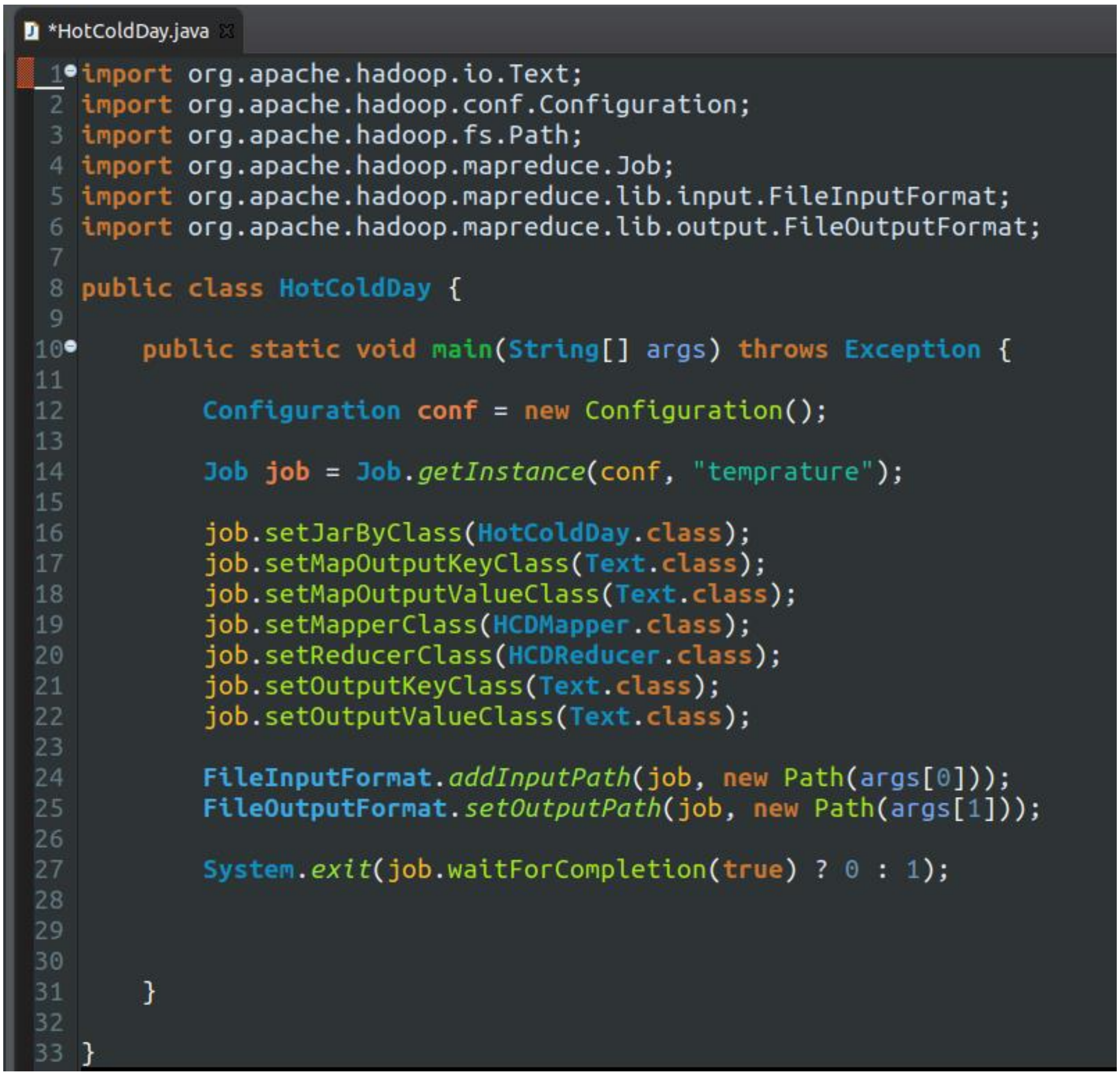
```
import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

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

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

        String line = value.toString();

        if(line.length() != 0) {

            String date = line.substring(6,14);

            float temp_max = Float.parseFloat(line.substring(41,47).trim());

            float temp_min = Float.parseFloat(line.substring(48,55).trim());

            if(temp_max > 30.0) {

                context.write(new Text(date), new Text("Hot Day "+temp_max));
            }


            if(temp_min < 10.0) {
                context.write(new Text(date), new Text("Cold Day "+temp_min));
            }


        }
    }

}
```

Reducer class :


HCDReducer.java

```

import java.io.IOException;
import java.util.Iterator;

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

public class HCDReducer extends Reducer<Text, Text, Text, Text>{

    public void reduce(Text key, Iterator<Text> values, Context context) throws IOException, InterruptedException{

        context.write(key, new Text(values.next().toString()));

    }

}
```

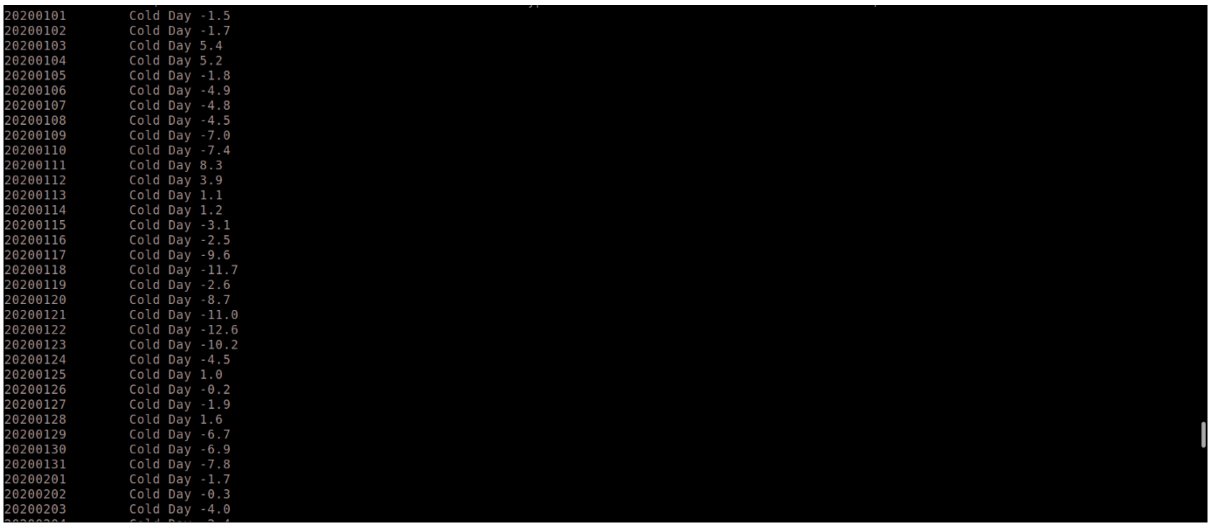
Main class :



*HotColdDay.java

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

public class HotColdDay {

    public static void main(String[] args) throws Exception {

        Configuration conf = new Configuration();

        Job job = Job.getInstance(conf, "temprature");

        job.setJarByClass(HotColdDay.class);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(Text.class);
        job.setMapperClass(HCDMapper.class);
        job.setReducerClass(HCDReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

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

        System.exit(job.waitForCompletion(true) ? 0 : 1);



    }

}
```

Output :-



```
20200101        Cold Day -1.5
20200102        Cold Day -1.7
20200103        Cold Day 5.4
20200104        Cold Day 5.2
20200105        Cold Day -1.8
20200106        Cold Day -4.9
20200107        Cold Day -4.8
20200108        Cold Day -4.5
20200109        Cold Day -7.0
20200110        Cold Day -7.4
20200111        Cold Day 8.3
20200112        Cold Day 3.9
20200113        Cold Day 1.1
20200114        Cold Day 1.2
20200115        Cold Day -3.1
20200116        Cold Day -2.5
20200117        Cold Day -9.6
20200118        Cold Day -11.7
20200119        Cold Day -2.6
20200120        Cold Day -8.7
20200121        Cold Day -11.0
20200122        Cold Day -12.6
20200123        Cold Day -10.2
20200124        Cold Day -4.5
20200125        Cold Day 1.0
20200126        Cold Day -0.2
20200127        Cold Day -1.9
20200128        Cold Day 1.6
20200129        Cold Day -6.7
20200130        Cold Day -6.9
20200131        Cold Day -7.8
20200201        Cold Day -1.7
20200202        Cold Day -0.3
20200203        Cold Day -4.0
```