# Breaking down MapReduce Concepts

## 1: Datatypes

Hadoop uses a wrapper around java datatypes. For eg: IntWritable is used for declaring an integer instead of int.

Similarly, Conversion needed from Java to Hadoop:
* byte -> ByteWritable
* short -> ShortWritable
* long -> LongWritable
* float -> FloatWritable
* double -> DoubleWritable
* boolean -> BooleanWritable
* string  -> Text

Explore more dataTypes : https://hadoop.apache.org/docs/r2.6.2/api/org/apache/hadoop/io/
Also have a look at : http://stackoverflow.com/questions/19441055/why-does-hadoop-need-classes-like-text-or-intwritable-instead-of-string-or-integ

Using these hadoop datatypes are easy. Let us look at an example.

1* To create a IntWritable from a int.
Method: public IntWritable(int value)
Eg: IntWritable var1 = new IntWritable(5);

2* To set value to a declared IntWritable variable.
Method: public void set (int value)
Eg: IntWritable var1 = new IntWritable();
    var1.set(5);

3* To get the value stored in the IntWritable variable.
Method: public int get()
Eg: var1.get(); // output will be 5;

Explore more methods at:
https://hadoop.apache.org/docs/r2.6.2/api/org/apache/hadoop/io/IntWritable.html

## 2: Understanding the mapreduce components

Note: Start reading from the main method for better understanding.

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

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.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

/* The following class has the implementation of Mapper,Combiner,Reducer and the main method. For simplicity,
we will use the same class for Reducer and Combiner.
*/
public class WordCount {

/* Extend the custom class with Mapper<IpKey, IpValue, OpKey, OpValue >
   IpKey ->InputKey to mapper class (Object in most cases)
   IpValue ->InputValue to mapper class (Text in most cases)
   OpKey ->OutputKey from mapper class (Will be the InputKey for Reducer)
   OpValue ->OutputValue from mapper class(Will be InputValue for Reducer)
*/
  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    //Create new IntWritable object
    private final static IntWritable one = new IntWritable(1);

    //Create new Text object
    private Text word = new Text();

    /*User logic is placed inside map function.
      Output (Key,value) pair is written to context in every stage,
      context facilitates the data movement from stage to another stage.
      Eg: from mapper class to reducer class
    */
    public void map(Object key, Text value, Context context)
        throws IOException, InterruptedException {

     /*Splits strings based on a token. Words separated by spaces in a sentence are divided into an array of words
here. Iterate through the array of words and assign value '1'to every word. Here word is the key and 1 is the value
for the corresponding key.
     */
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());

        // context.write(key,value)
        context.write(word, one);
      }
    }
  }
```

```
62
63  /* Extend the custom class with Reducer<IpKey, IpValue, OpKey, OpValue >
64    IpKey ->InputKey to reducer class (OutputKey from mapper class)
65    IpValue ->InputValue to reducer class (OuputValue from mapper class)
66    OpKey ->OutputKey from reducer class (Will be written to file)
67    OpValue ->OutputValue from reducer class(Will be written to file)
68  */
69    public static class IntSumReducer
70        extends Reducer<Text,IntWritable,Text,IntWritable>{
71     private IntWritable result = new IntWritable();
72
73     /*User logic is placed inside reducefunction.
74       Output (Key,value) pair is written to context.
75       DataType of key in reduce is Text because outputKey from mapper is a Text. So the dataType must be
76  changed according to the mapper output.
77       DataType of Value is Iterable<IntWritable>because outputValue from mapper is a IntWritable. As a particular
78  key can have multiple values,
79  we adopt Iterable<InputValue DataType>as the standard.
80     */
81     public void reduce(Text key, Iterable<IntWritable>values,
82                        Context context
83                        ) throws IOException, InterruptedException {
84       int sum = 0;
85
86       //For every key, sum the value
87       for (IntWritable val : values) {
88         sum += val.get();
89       }
90       result.set(sum);
91       context.write(key, result);
92     }
93    }
94
95
96
97  public static void main(String[] args) throws Exception {
98     //Create an object of org.apache.hadoop.conf.Configuration
99     Configuration conf = new Configuration();
100
101    /* Create an object of org.apache.hadoop.mapreduce.Job.
102       Creates a new Jobwith a given jobName. The Job makes a copy of theConfiguration so that any necessary
103  internal modifications do not reflect on the incoming parameter.
104    */
105    Job job = Job.getInstance(conf, "word count");
106
107    // Set the class name in which main method resides.
108    job.setJarByClass(WordCount.class);
109    // Set the mapper class name
110    job.setMapperClass(TokenizerMapper.class);
111
112    //Set the combiner class name
113    job.setCombinerClass(IntSumReducer.class);
114
115    //Set the reducer class name
116    job.setReducerClass(IntSumReducer.class);
117
118    /*Output from reducer is the final result and Output is Key,Value pair.
119      Inform Hadoop about the dataType of Key and Value from reducer
120    */
121    job.setOutputKeyClass(Text.class);
122    job.setOutputValueClass(IntWritable.class);
123
124
125
```

```
126   /* The following command is used to run a mapreduce program.
127      hadoop jar wc.jar WordCount input_dir output_dir
128      hadoop jar <jar name><pgm class name><input path><output path>
129      The input and output path to the dataset are set as below
130   */
131   FileInputFormat.addInputPath(job, new Path(args[0]));
132   FileOutputFormat.setOutputPath(job, new Path(args[1]));
133
134   /* Wait for the program execution to complete.
135      Exit the program after job execution.
136   */
137   System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

## 3: Understanding the mapreduce workflow with an example