Basic HADOOP API (1.x or 0.20.x)

- **Package org.apache.hadoop.mapreduce**
- **Class Mapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT>**
  - void setup(Mapper.Context context)
  - void cleanup(Mapper.Context context)
  - void map(KEYIN key, VALUEIN value, Mapper.Context context)
  - output is generated by invoking context.collect(key, value);
- **Class Reducer<KEYIN, VALUEIN, KEYOUT, VALUEOUT>**
  - void setup(Reducer.Context context)
  - void cleanup(Reducer.Context context)
  - void reduce(KEYIN key, Iterable<VALUEIN> values, Reducer.Context context)
  - output is generated by invoking context.collect(key, value);
- **Class Partitioner<KEY, VALUE>**
  - abstract int getPartition(KEY key, VALUE value, int numPartitions)

JOB

- Represents a packaged Hadoop job for submission to cluster
- Need to specify input and output paths
- Need to specify input and output formats
- Need to specify mapper, reducer, combiner, partitioner classes
- Need to specify intermediate/final key/value classes
- Need to specify number of reducers (but not mappers, why?)
- Don’t depend of defaults!

Basic HADOOP main (1.x or 0.20.x)

```
public static void main(String[] args) throws Exception
{
        Configuration conf = new Configuration();
        Job job = new Job(conf, "wordcount");
        job.setJarByClass(WordCount.class);

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

        job.setMapperClass(NewMapper.class);
        job.setReducerClass(NewReducer.class);

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

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

Basic HADOOP Data Types (1.x or 0.20.x)

- **Package org.apache.hadoop.io**

interface Writable

↑

interface WritableComparable<T>

↑

IntWritable
LongWritable
Text

Defines a de/serialization protocol

Any key or value type in the Hadoop Map-Reduce framework implements this interface

WritableComparables can be compared to each other, typically via Comparators

Any type which is to be used as a key in the Hadoop Map-Reduce framework should implement this interface

Concrete classes for common data types

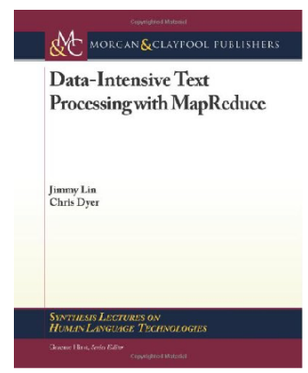
Complex HADOOP Data Types

- **Quick & Dirty way**
  - Encode key and value as Text object with custom separator
  - Example: ("blue", 14) becomes "blue_14" or "blue$14)
  - Use regular expressions or split() to extract data
  - Good for rapid prototyping, bad for performance
- **Standard way**
  - Define a custom implementation of WritableComparable<T>
    - Must implement
      - public void write(DataOutput out) throws IOException
      - public void readFields(DataInput in) throws IOException
      - public int compareTo(T o)
    - Should implement
      - public int hashCode()
      - public boolean equals(Object obj)
  - Good for performance, bad for rapid prototyping

Hello World in Hadoop (I)

```
1: class Mapper
2:     method Map(docid a, doc d)
3:         for all term t ∈ doc d do
4:             Emit(term t, count 1)

1: class Reducer
2:     method Reduce(term t, counts [c1, c2, ...])
3:         sum ← 0
4:         for all count c ∈ counts [c1, c2, ...] do
5:             sum ← sum + c
6:         Emit(term t, count sum)
```



Hello World in Hadoop (II)

```
public static class MyMapper extends Mapper<Object, Text, Text, IntWritable>
{
   private final static IntWritable ONE = new IntWritable(1);
   private Text WORD = new Text();

   @Override
   public void map(Object 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);
      }
   }
}
```

Hello World in Hadoop (III)

```
public static class MyReducer 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 += val.get();
      }
      result.set(sum);
      context.write(key, result);
   }
}
```