Karlsruhe Institute of Technology

Hadoop tutorial

MapReduce concepts

A. Hammad, A. García | September 7, 2011

STEINBUCH CENTRE FOR COMPUTING (SCC)

MapReduce I
Streaming

What is Hadoop?

The Apache Hadoop project develops **open-source software for reliable, scalable, distributed computing**.

In a nutshell
Hadoop provides: a reliable shared storage and analysis system.

The **storage** is provided by **HDFS**
The **analysis** by **MapReduce**.

What is MapReduce?

MapReduce is a programming model for data processing.

Hadoop/MapReduce Terminology

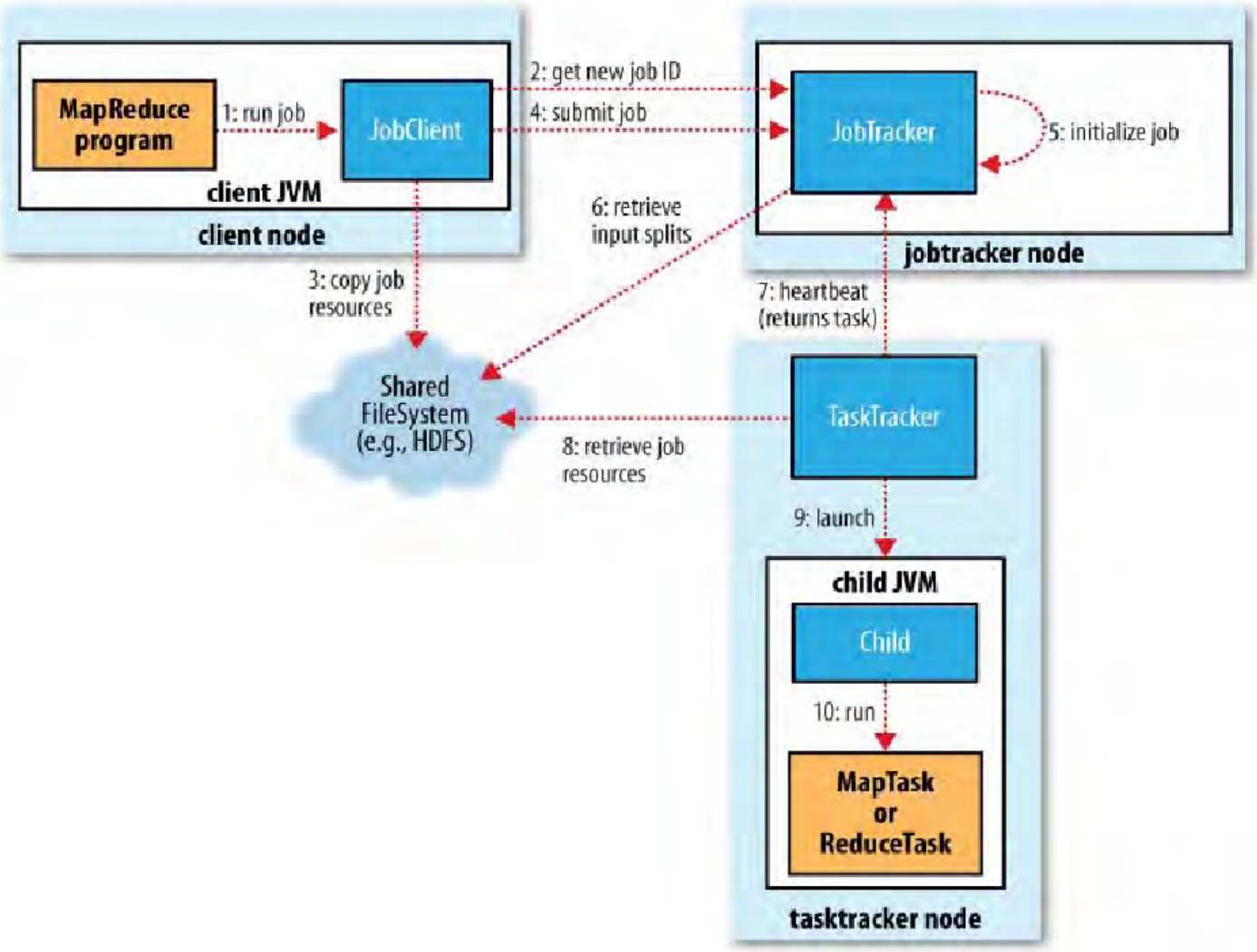
- MR Job, Streaming
- MapTask, ReduceTask
- map, reduce, Driver
- Counters, Property

- Data locality
- InputSplit
- Key/Value pairs
- Shuffle

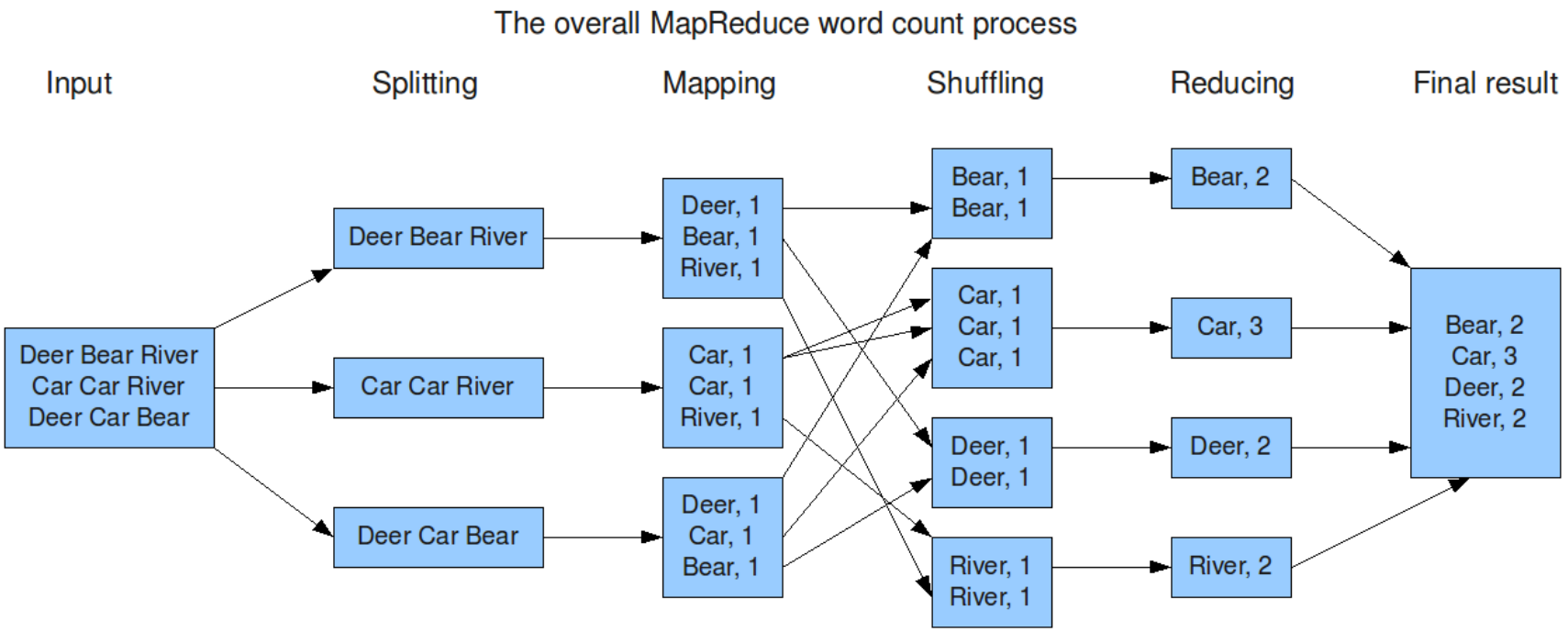
- NameNode, JobTraker
- Datanode, TaskTraker
- HDFS, Block

- File InputFormat
- File OutputFormat
- Writables

MapReduce Job Flow



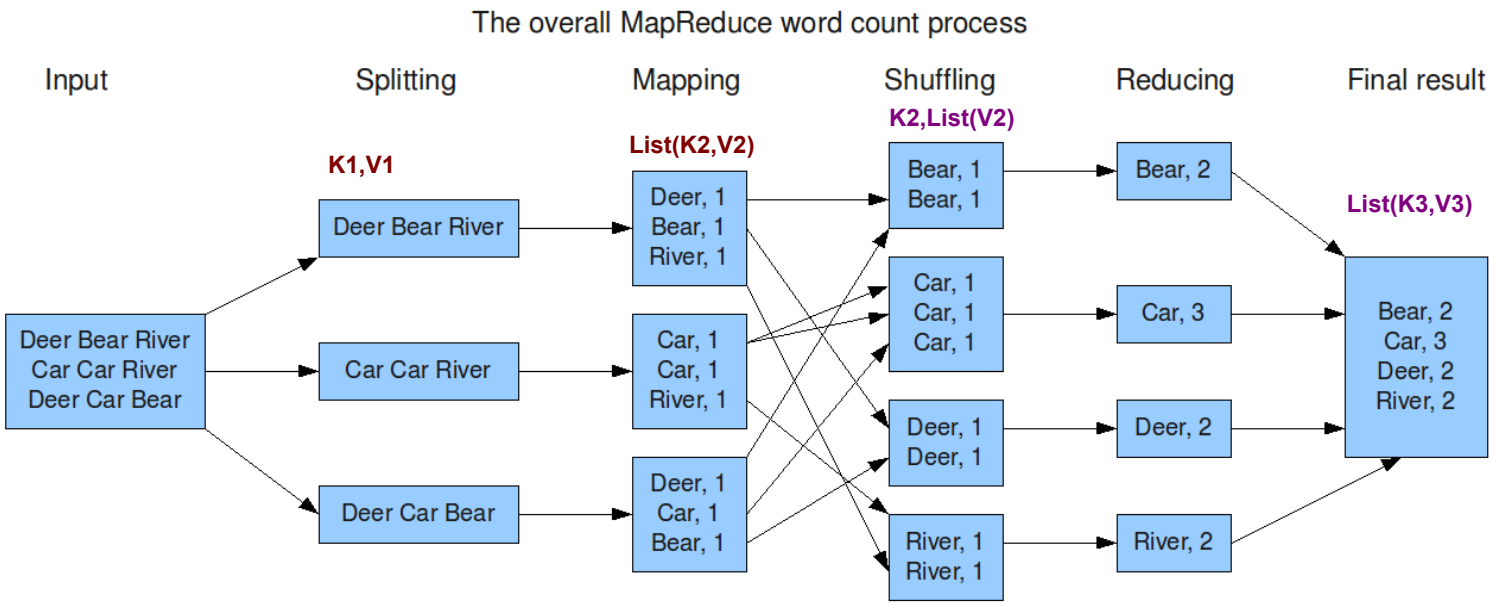
A MapReduce Example



The MapReduce paradigm

map: (K1, V1) → list(K2, V2)

reduce: (K2, list(V2)) → list(K3, V3)



Exercise 1: Hadoop MR Streaming

Hadoop Streaming: An API to MapReduce to write map and reduce functions in languages other than Java.

It uses STDIN to **read text data line-by-line** and write to STDOUT.

Map input data is passed to your map function.
A map **key-value pair** is written as a single **tab-delimited** line to STDOUT.

The reduce function input is a tab-separated key-value pair passed over STDIN, and writes its results to STDOUT.

Exercise 1: Hadoop MR Streaming

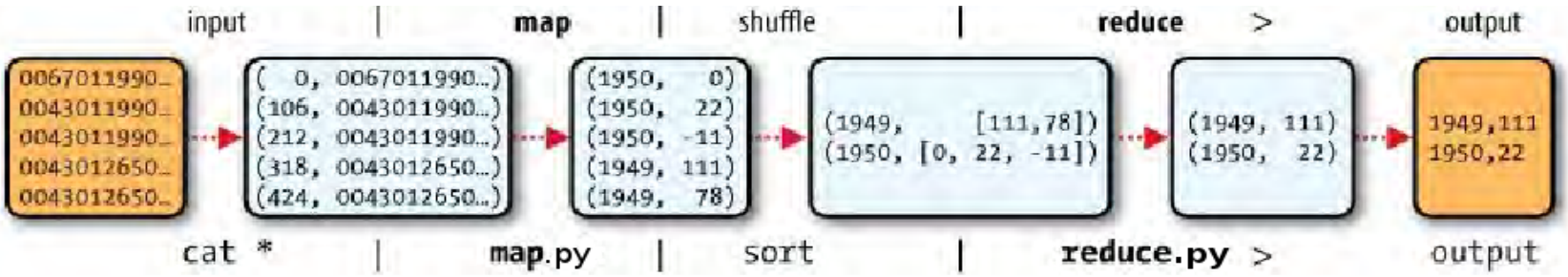
What's the highest recorded temperature for each year in the dataset?

0067011990999991950051507004+68750+023550FM-12+038299999V0203301N00671220001CN9999999N9+00001+99999...
0043012650999991949032418004+62300+010750FM-12+048599999V0202701N00461220001CN0500001N9-00785+99999..

Weather data set Format:

Position	value	Description
1-10	0067011991	USAF weather station identifier
11-15	99999	WBAN weather station identifier
16-23, (**Year: 16-19**)	19500515	observation date
		observation time
	.	latitude (degrees x 1000)
	.	longitude (degrees x 1000)
	.	elevation (meters)
	.	wind direction (degrees)
		quality code
		sky ceiling height (meters)
		quality code
		visibility distance (meters)
		quality code
88-92	**+0000**	**air temperature** (**degrees Celsius x 10**)
93	**0,1,4,5 or 9**	**quality code**
		dew point temperature (degrees Celsius x 10)
		quality code
		atmospheric pressure (hectopascals x 10)
		quality code

The map and reduce



- Testing without Hadoop:

```
$ cat sample.txt | ./map.py | sort | ./reduce.py
```

- Testing with Hadoop Streaming:

```
$ hdoop jar /usr/lib/hadoop/contrib/streaming/hadoop-streaming-0.20.2-cdh3u0.jar \
        -input input/ncdc/sample.txt \
        -output output_ex1 \
        -mapper ./map.py \
        -reducer ./reduce.py \
        -file ./map.py \
        -file ./reduce.py
```

MapReduce II
Developing MR programs in Java

MapReduce API

For	Imports	Classes&Interfaces
Configuration	org.apache.hadoop.conf.*	Configuration, Configured, Tool
MR Job, Formats	org.apache.hadoop.mapred.*	JobClient, JobConf, MapReduceBase, Mapper, Reducer, FileInputFormat, FileOutputFormat, TextInputFormat, TextOutputFormat,... OutputCollector, Reporter
Data types	org.apache.hadoop.io.*	Text, LongWritable, FloatWritable, ByteWritable,...
File system	org.apache.hadoop.fs.*	FileSystem, FSDataInputStream, FSDataOutputStream, Path, FileStatus ...
Utilities	org.apache.hadoop.utils.* org.apache.hadoop.IOUtils.*	ToolRunner, copyBytes, ReadFully
Native Java	java.io.IOException; java.util.Iterator, ...	

MapReduce data types: Writables

import org.apache.hadoop.io.*

Writable wrapper classes for Java primitives

Class	Size in bytes	Description
BooleanWritable	1	Wrapper for a standard Boolean variable
ByteWritable	1	Wrapper for a single byte
DoubleWritable	8	Wrapper for a Double
FloatWritable	4	Wrapper for a Float
IntWritable	4	Wrapper for a Integer
LongWritable	8	Wrapper for a Long
Text	2GB	Wrapper to store text using the unicode UTF8 format
NullWritable		Placeholder when the key or value is not needed
*Your*Writable		Implement the Writable Interface for a value or WritableComparable<T> for a key

MapReduce InputFormat

import org.apache.hadoop.mapred.*

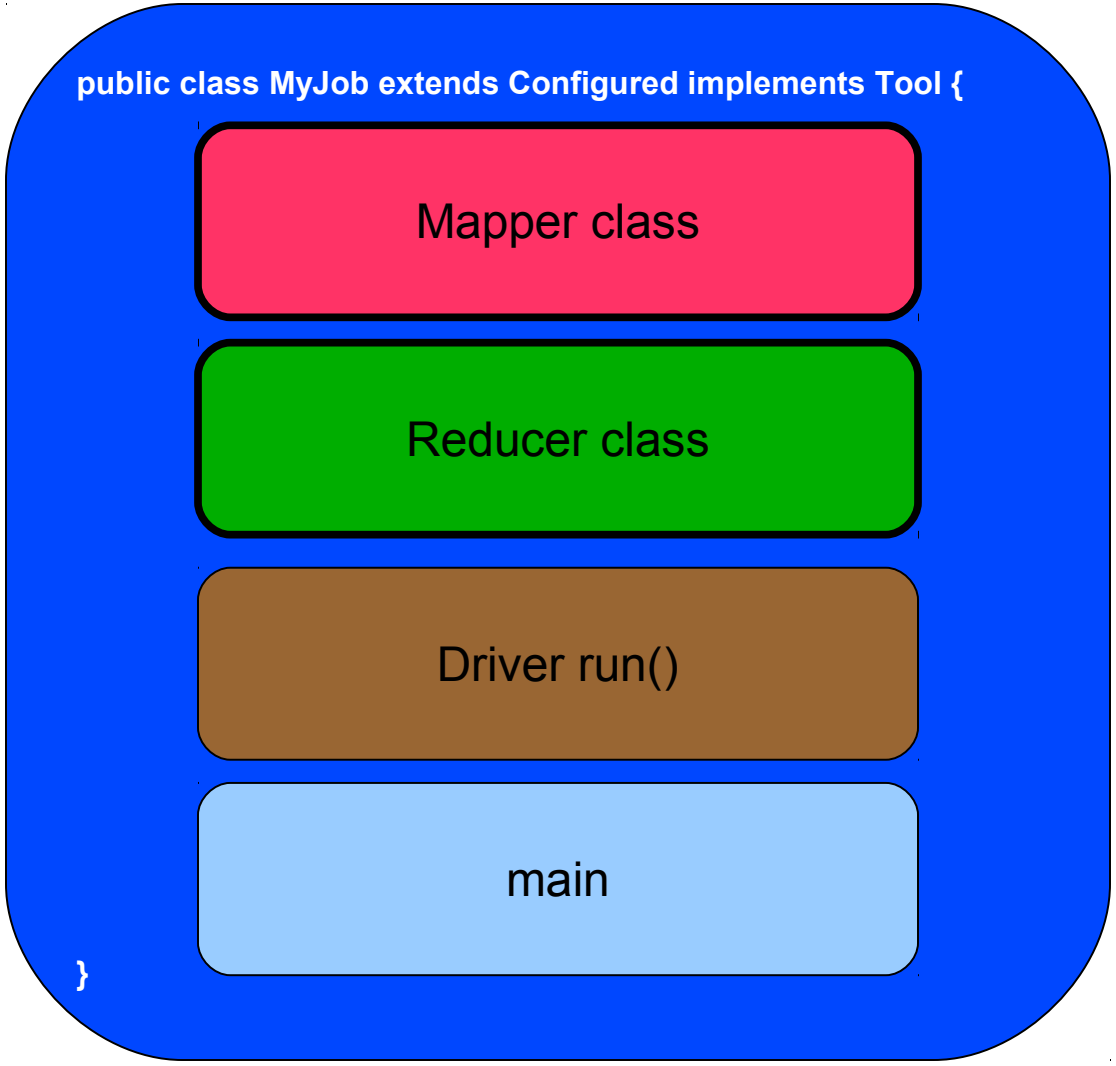
Input Format	Description
TextInputFormat	Each line in text file is a record. **key**: LongWritable, **Offset** of the line **value**: Text, content of the line
KeyValueTextInputFormat	Each line is a record. First separator divides each line. Separator set by **key.value.separator.in.input.line** property, default is tab character (\t). **key**: Text, anything before the separator **value**: Text, everything after the separator
SequenceFileInputFormat<K,V>	An InputFormat for reading sequence files. A sequence file is a Hadoop-specific compressed binary file format. **key**: K (user defined) **value**: V (user defined)
NLineInputFormat	Like TextInputFormat, but each split is guaranteed to have exactly N lines. Set by **mapred.line.input.format.linespermap** property **key**: LongWritable **value**: Text

MapReduce OutputFormat

import org.apache.hadoop.mapred.*

TextOutputFormat<K,V>	Writes each record as a line of text. Keys and values are written as strings and separated by a tab (\t) character, which can be changed in the **mapred.textoutputformat.separator** property.
SequenceFileOutputFormat<K,V>	Writes the key/value pairs in sequence file format. Works in conjunction with SequenceFileInputFormat.
NullOutputFormat<K,V>	Outputs nothing

MapReduce program components



Skeleton of a MapReduce program

```
public class WordCount extends Configured implements Tool {

  public static class MyMapper extends MapReduceBase
    implements 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,
        OutputCollector<Text, IntWritable> output,
        Reporter reporter) throws IOException {

      String line = value.toString();
      StringTokenizer itr = new StringTokenizer(line);
      while (itr.hasMoreTokens()) {
            word.set(itr.nextToken());
            output.collect(word, one);
      }
    }
  }
```

Skeleton of a MapReduce program

```
public static class MyReducer extends MapReduceBase
   implements Reducer<Text, IntWritable, Text, IntWritable> {

   public void reduce(Text key, Iterator<IntWritable> values,
      OutputCollector<Text, IntWritable> output,
      Reporter reporter) throws IOException {

      int sum = 0;
      while (values.hasNext()) {
            sum += values.next().get();
      }
      output.collect(key, new IntWritable(sum));
   }
}
```

Skeleton of a MapReduce program

```
public int run(String[] args) throws Exception {
    Configuration conf = getConf();
    JobConf job = new JobConf(conf, MyJob.class);

    Path in = new Path(args[0]);
    Path out = new Path(args[1]);
    FileInputFormat.setInputPaths(job, in);
    FileOutputFormat.setOutputPath(job, out);

    job.setJobName("Word Counter");
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    job.setMapperClass(MyMapper.class);
    job.setReducerClass(MyReducer.class);

    JobClient.runJob(job);
    return 0;
}

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

Exercise 2: MapReduce Job in Java

Implement the Map, Reduce and the driver functions to find the max temperature.

```
# Compilation
javac -classpath /usr/lib/hadoop/hadoop-core.jar  -d myclasses  MyJob.java

# Archiving
jar -cvf  myjob.jar  -C myclasses/ .

# Run
hadoop jar myjob.jar MyJob  /share/data/gks2011/input/sample.txt \
                            /tmp/gks/gs0xx/myOutput
```

MapReduce III
Counters

MapReduce Counters

Counters show statistics about the MR job and the Data.

Hadoop maintains built-in Counters as seen by your jobs logging output.

Hadoop allows defining your own Counters to better analyze your data.

For instance to count the number of ***bad records***.

How to define your own Counters with Java?

MapReduce user-defined Counters

```
public class MyJobWithCounters extends Configured implements Tool {
  // Counters
  enum Temperature { MISSING,  MALFORMED  }
```

Counters are incremented by the ***Reporter.incrCounter()*** method.
The Reporter object is passed to the map() and reduce() methods.

public void incrCounter(Enum key, long amount);

Example: reporter.incrCounter(Temperature.MISSING, 1);

The Dynamic counter, No *enum* needed!

public void incrCounter(String group, String counter, long amount)

Example: reporter.incrCounter("TemperatureQuality", parser.getQuality(), 1);

Exercise 3

Karlsruhe Institute of Technology

Implementation of user-defined Counters