MapReduce Types and Formats

Outline

- MapReduce Types
- Input Formats
- Output Formats

MapReduce Types

- General form

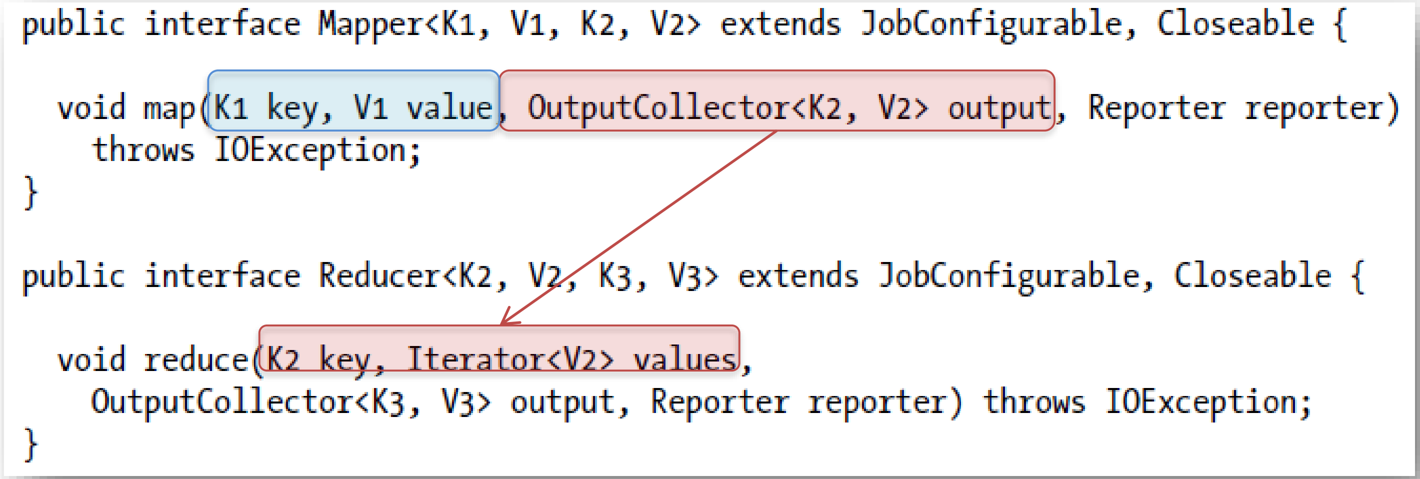
≠

map: (K1, V1) → list(K2, V2)

=

reduce: (K2, list(V2)) → list(K3, V3)

- Java interface

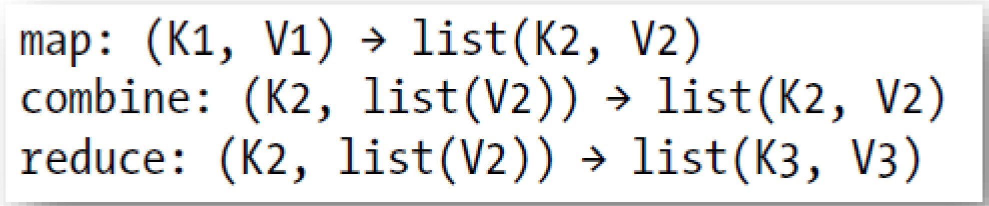


```
public interface Mapper<K1, V1, K2, V2> extends JobConfigurable, Closeable {

  void map(K1 key, V1 value, OutputCollector<K2, V2> output, Reporter reporter)
    throws IOException;
}

public interface Reducer<K2, V2, K3, V3> extends JobConfigurable, Closeable {

  void reduce(K2 key, Iterator<V2> values,
    OutputCollector<K3, V3> output, Reporter reporter) throws IOException;
}
```

  – ***OutputCollector*** emitts key-value pairs
  – ***Reporter*** updates counters and status

MapReduce Types

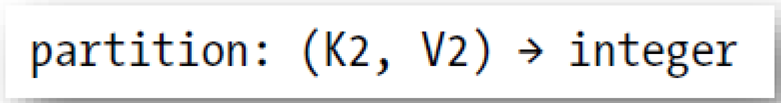
- Combine function



```
map: (K1, V1) → list(K2, V2)
combine: (K2, list(V2)) → list(K2, V2)
reduce: (K2, list(V2)) → list(K3, V3)
```

  - The same form as the reduce function, except its output types
  - Output type is the same as Map
  - Often the combine and reduce functions are the same

- Partition function



```
partition: (K2, V2) → integer
```

  - Input intermediate key and value types
  - Returns the partition index

MapReduce Types

- Input types are set by the input format

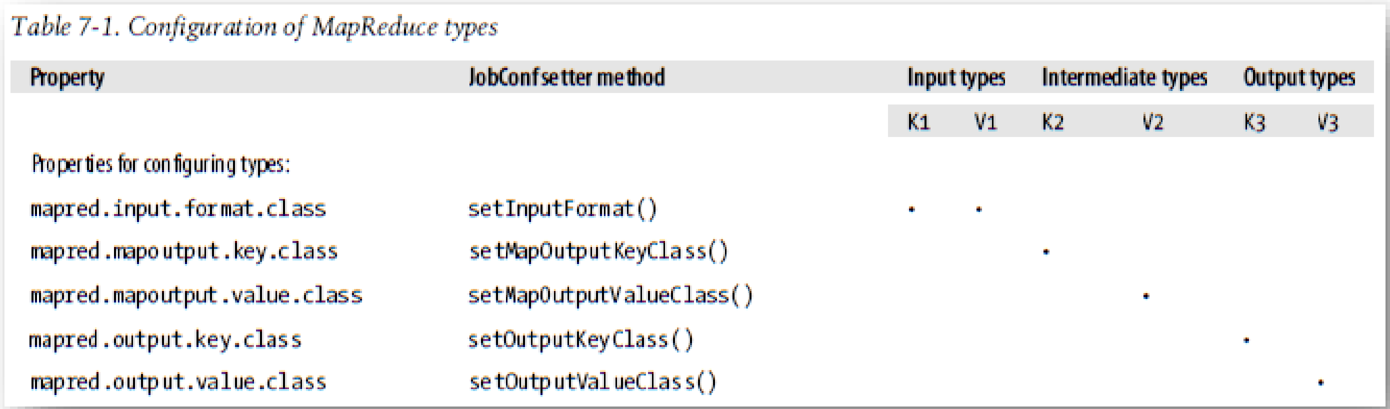


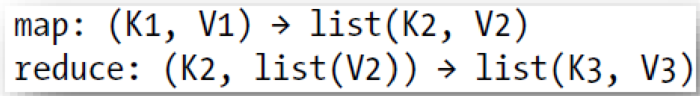
Table 7-1. Configuration of MapReduce types

Property	JobConf setter method	Input types		Intermediate types		Output types	
		K1	V1	K2	V2	K3	V3
Properties for configuring types:							
`mapred.input.format.class`	`setInputFormat()`	•	•				
`mapred.mapoutput.key.class`	`setMapOutputKeyClass()`			•			
`mapred.mapoutput.value.class`	`setMapOutputValueClass()`				•		
`mapred.output.key.class`	`setOutputKeyClass()`					•	
`mapred.output.value.class`	`setOutputValueClass()`						•

Ex) **setInputFormat(TextInputFormat.class)**

  - Generate Key type: **LongWritable**, Value type: **Text**

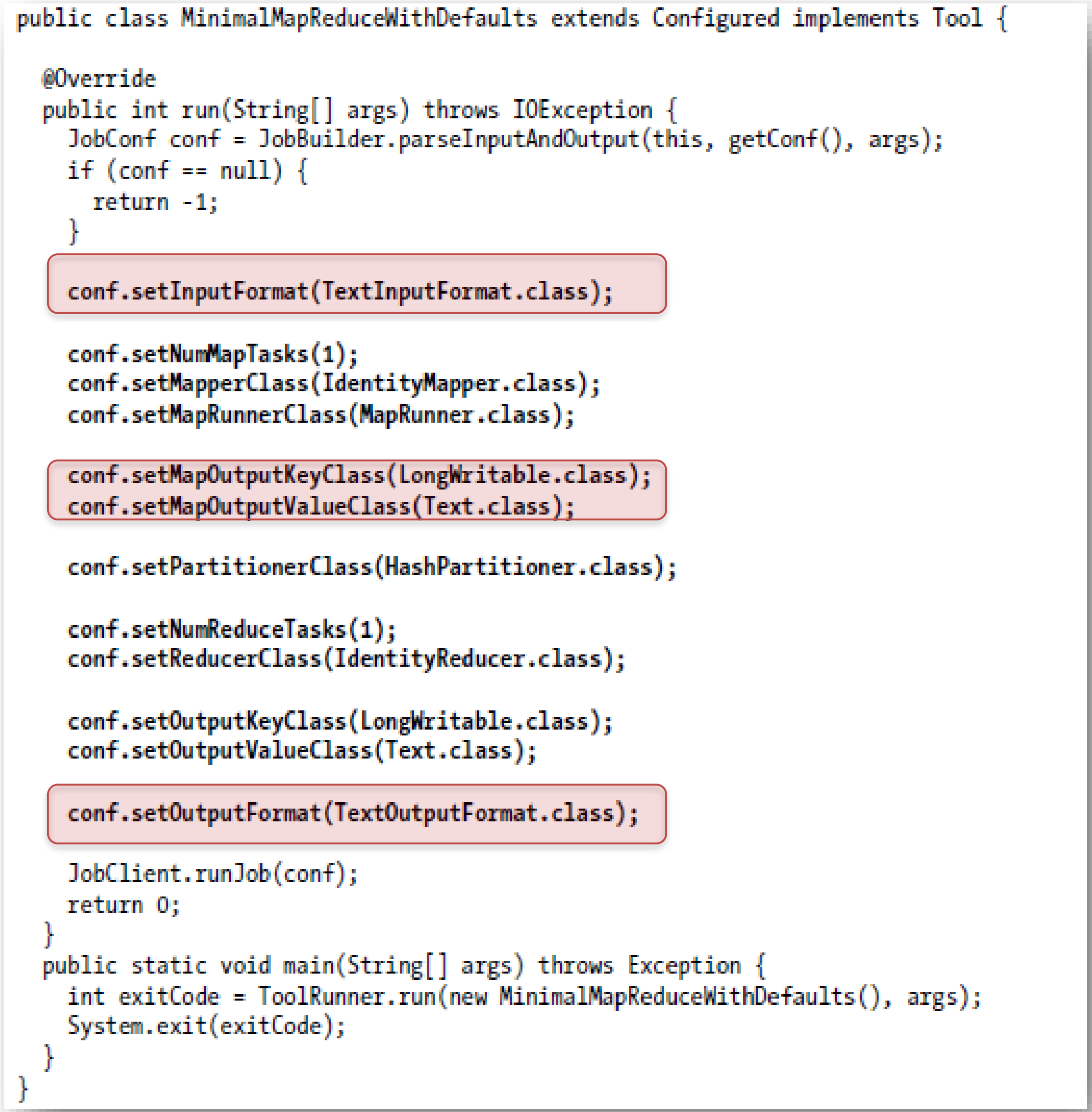
- Other types are set explicitly by calling the methods on the **JobConf**
  - Ex) JobConf conf;   conf.setMapOutputKeyClass(Text.class)
- Intermediate types are also set as the final output types by default
  - Just need to call **setOutputKeyClass()** if K2 and K3 are the same



```
map: (K1, V1) → list(K2, V2)
reduce: (K2, list(V2)) → list(K3, V3)
```

MapReduce Types

The Default MapReduce Job



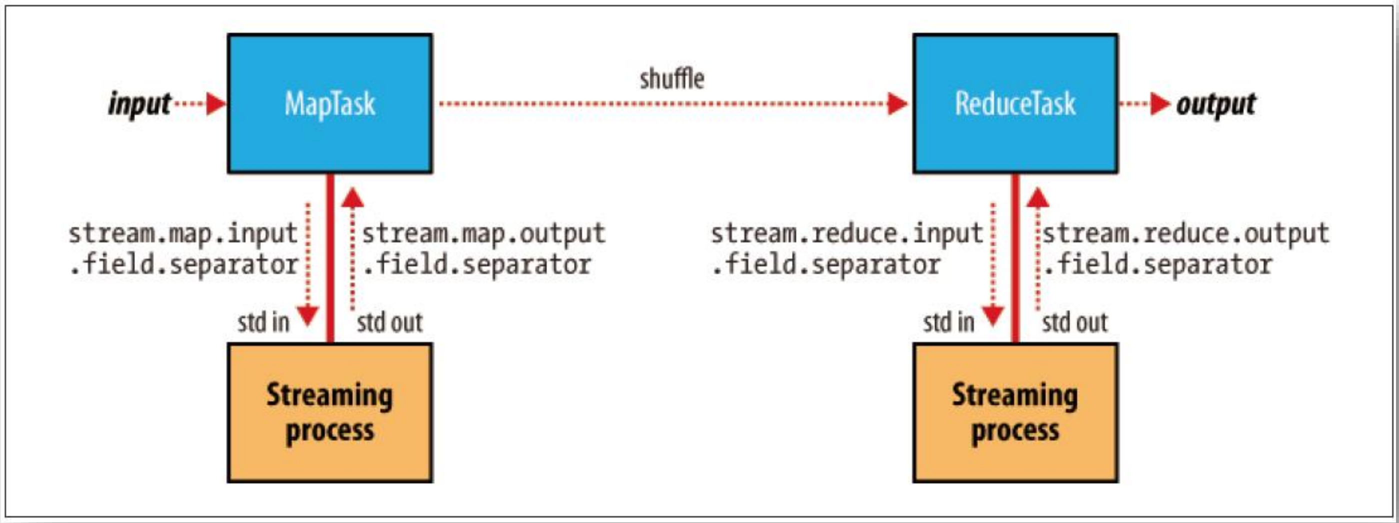
```
public class MinimalMapReduceWithDefaults extends Configured implements Tool {

  @Override
  public int run(String[] args) throws IOException {
    JobConf conf = JobBuilder.parseInputAndOutput(this, getConf(), args);
    if (conf == null) {
      return -1;
    }

    conf.setInputFormat(TextInputFormat.class);

    conf.setNumMapTasks(1);
    conf.setMapperClass(IdentityMapper.class);
    conf.setMapRunnerClass(MapRunner.class);

    conf.setMapOutputKeyClass(LongWritable.class);
    conf.setMapOutputValueClass(Text.class);

    conf.setPartitionerClass(HashPartitioner.class);

    conf.setNumReduceTasks(1);
    conf.setReducerClass(IdentityReducer.class);

    conf.setOutputKeyClass(LongWritable.class);
    conf.setOutputValueClass(Text.class);

    conf.setOutputFormat(TextOutputFormat.class);

    JobClient.runJob(conf);
    return 0;
  }
  public static void main(String[] args) throws Exception {
    int exitCode = ToolRunner.run(new MinimalMapReduceWithDefaults(), args);
    System.exit(exitCode);
  }
}
```

- Default Input Format
  - TextInputFormat
    - LongWritable (Key)
    - Text (Value)
- setNumMapTasks(1)
  - Does not set the number of map tasks to one
    - 1 is a hint

Choosing the Number of Reducers

- The optimal number of reducers is related to the total number of available reducer slots
  - Total number of available reducers =
    Total nodes * mapred.tasktracker.reduce.tasks.maximum
- To have slightly fewer reducers than total slots
  - Tolerate a few failures without extending job execution time

Keys and values in Streaming

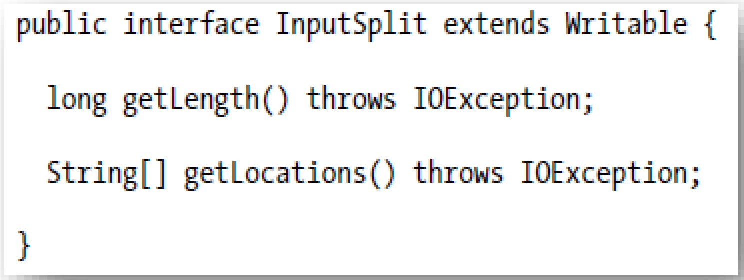

input
MapTask
shuffle
ReduceTask
output
stream.map.input .field.separator
stream.map.output .field.separator
stream.reduce.input .field.separator
stream.reduce.output .field.separator
std in
std out
std in
std out
Streaming process
Streaming process

- A streaming application can control the separator
  - Default separator: tab character
  - Separators may be configured independently for maps and reduces
  - The number of fields separated by itself to treat as the map output key
    - Set the first **n** fields in **stream.num.map.output.key.fields**
    - Ex) Output was **a,b,c** (and separator is a comma), **n**=2
      - Key: a,b   Value:c

Outline

- MapReduce Types
- Input Formats
  - Input Splits and Records
  - Text Input
  - Binary Input
  - Multiple Inputs
  - Database Input(and Output)
- Output Formats

Input Splits and Records



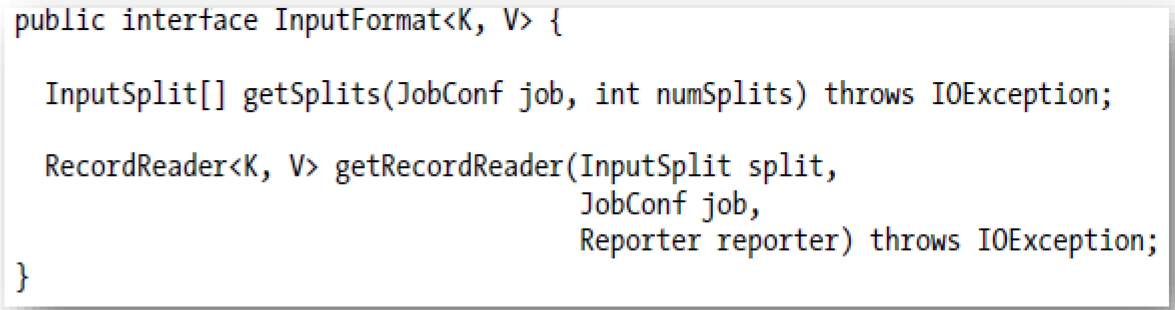
```
public interface InputSplit extends Writable {

  long getLength() throws IOException;

  String[] getLocations() throws IOException;

}
```

- InputSplit (org.apache.hadoop.mapred package)
  - A **chunk** of the input processed by a single map
  - Each split is divided into records
  - Split is just a reference to the data (Doesn’t contain the input data)

- Ordering the splits
  - To process the largest split first (minimize the job runtime)

Input Splits and Records - InputFormat

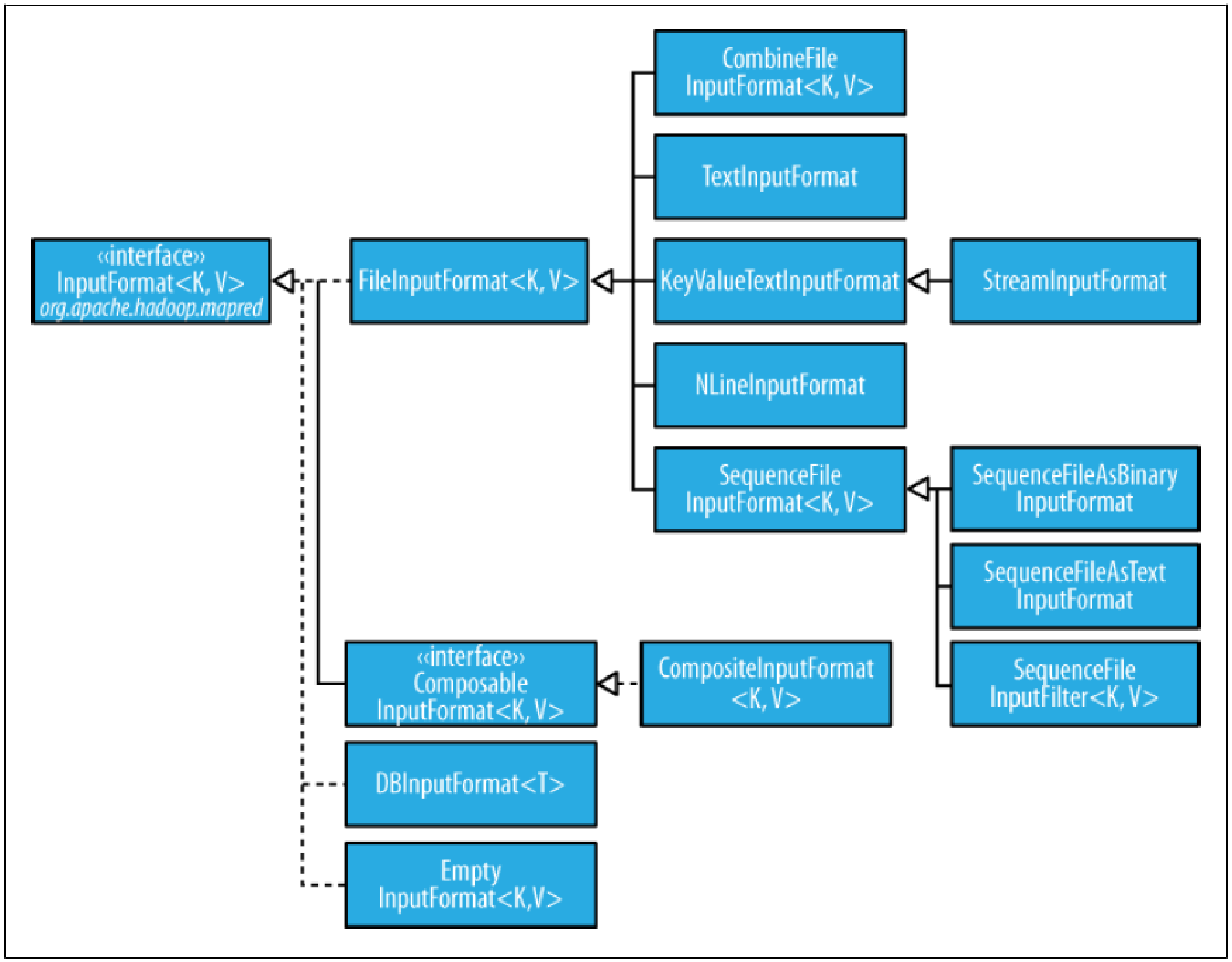
- Create the input splits, and dividing them into records
- **numSplits** argument of getSplits() method is a hint
  - InputFormat is free to return a different number of splits
- The client sends the calculated splits to the jobtracker
  - Schedule map tasks to process on the tasktrackers
- RecordReader
  - Iterate over records
  - Used by the map task to generate record key-value pairs



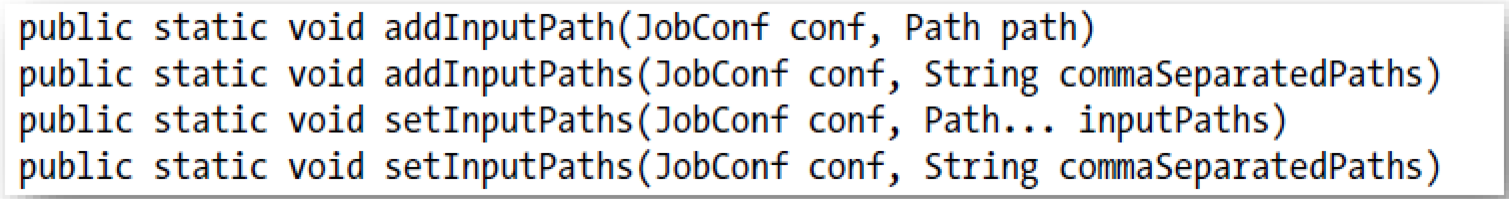
```
public interface InputFormat<K, V> {

  InputSplit[] getSplits(JobConf job, int numSplits) throws IOException;

  RecordReader<K, V> getRecordReader(InputSplit split,
                                     JobConf job,
                                     Reporter reporter) throws IOException;
}
```

Input Splits and Records - FileInputFormat

- The base class for all InputFormat that use files as their data source


«interface» InputFormat<K, V> org.apache.hadoop.mapred
FileInputFormat<K, V>
CombineFile InputFormat<K, V>
TextInputFormat
KeyValueTextInputFormat
StreamInputFormat
NLineInputFormat
SequenceFile InputFormat<K, V>
SequenceFileAsBinary InputFormat
SequenceFileAsText InputFormat
SequenceFile InputFilter<K, V>
«interface» Composable InputFormat<K, V>
CompositeInputFormat <K, V>
DBInputFormat<T>
Empty InputFormat<K,V>

Input Splits and Records - FileInputFormat



```
public static void addInputPath(JobConf conf, Path path)
public static void addInputPaths(JobConf conf, String commaSeparatedPaths)
public static void setInputPaths(JobConf conf, Path... inputPaths)
public static void setInputPaths(JobConf conf, String commaSeparatedPaths)
```

- FileInputFormat offers 4 static methods for setting a JobConf's input paths
  - addInputPath() and addInputPaths()
    - Add a path or paths to the list of inputs
    - Can call these methods repeatedly
  - setInputPaths()
    - Set entire list of paths in one time (Replacing any paths that were set in previous calls)
  - A path may represent
    - A file
    - A directory (consider all files in the directory as input)
      - Error when subdirectory exists (solved by glob or filter)
    - A collection of files and directories by using a glob

Input Splits and Records - FileInputFormat

- Filters
  - Use FileInputFormat as a default filter
    - Exclude hidden files
  - Use setInputPathFilter() method
    - Act in addition to the default filter
    - Refer page 61

Input Splits and Records – FileInputFormat input splits

- **FileInputFormat** splits only large files that larger than an HDFS block
  - Normally the split size is the size of an HDFS block

- Possible to control the split size
  - Effect maximum split size: The maximum size is less than block size

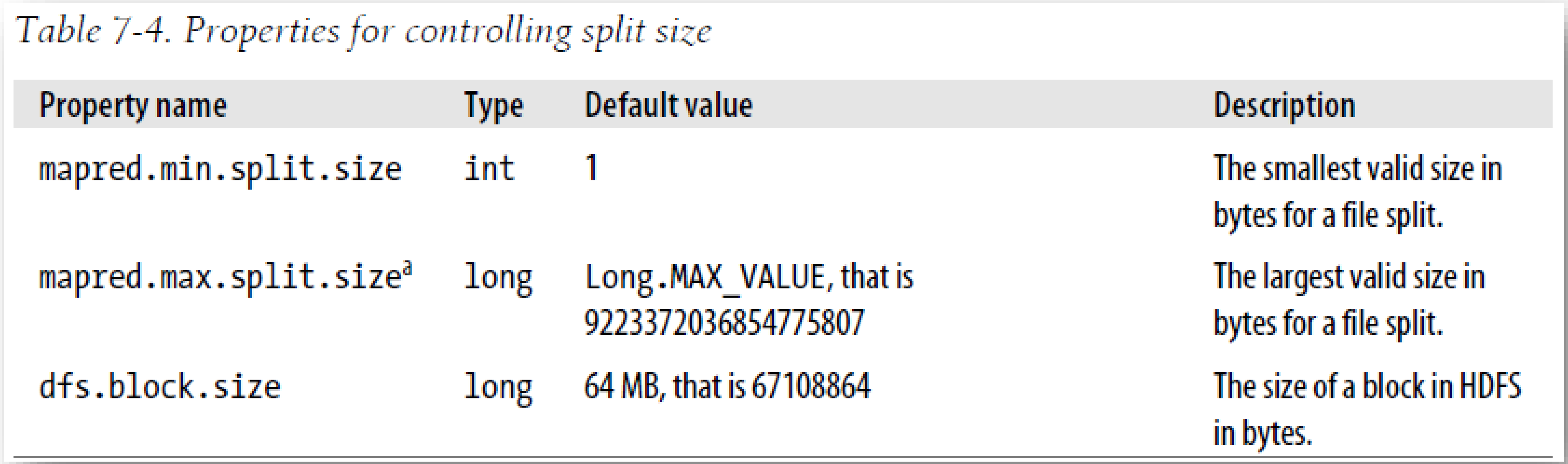


Table 7-4. Properties for controlling split size

Property name	Type	Default value	Description
`mapred.min.split.size`	`int`	1	The smallest valid size in bytes for a file split.
`mapred.max.split.size`[a]	`long`	`Long.MAX_VALUE`, that is 9223372036854775807	The largest valid size in bytes for a file split.
`dfs.block.size`	`long`	64 MB, that is 67108864	The size of a block in HDFS in bytes.

Input Splits and Records – FileInputFormat input splits

- The split size calculation (computeSplitSize() method)

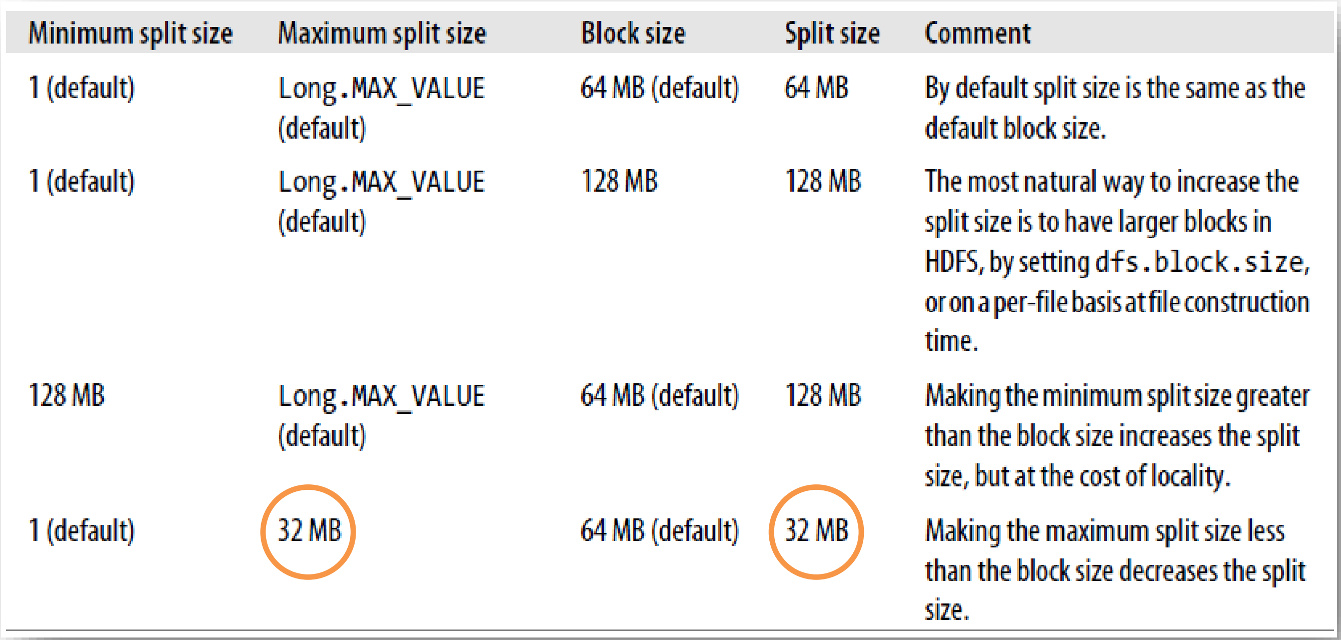
```
max(minimumSize, min(maximumSize, blockSize))
```

- By default

```
minimumSize < blockSize < maximumSize
```

  – Split size is blockSize

- Control the split size



Minimum split size	Maximum split size	Block size	Split size	Comment
1 (default)	`Long.MAX_VALUE` (default)	64 MB (default)	64 MB	By default split size is the same as the default block size.
1 (default)	`Long.MAX_VALUE` (default)	128 MB	128 MB	The most natural way to increase the split size is to have larger blocks in HDFS, by setting `dfs.block.size`, or on a per-file basis at file construction time.
128 MB	`Long.MAX_VALUE` (default)	64 MB (default)	128 MB	Making the minimum split size greater than the block size increases the split size, but at the cost of locality.
1 (default)	32 MB	64 MB (default)	32 MB	Making the maximum split size less than the block size decreases the split size.

Input Splits and Records – Small files and CombineFileInputFormat

- Hadoop works better with a small number of large files than a large number of small files
  - FileInputFormat generates splits that each split is all or part of a single file
  - Bookkeeping overhead with a lot of small input data
- Use CombineFileInputFormat to pack many files into splits
  - Designed to work well with small files
  - Take node and rack locality when packing blocks into split
  - Worth when already have a large number of small files in HDFS
- Avoiding the many small files is a good idea
  - Reduce the number of seeks
  - Merge small files into larger files by using a SequenceFile

Input Splits and Records – Preventing splitting

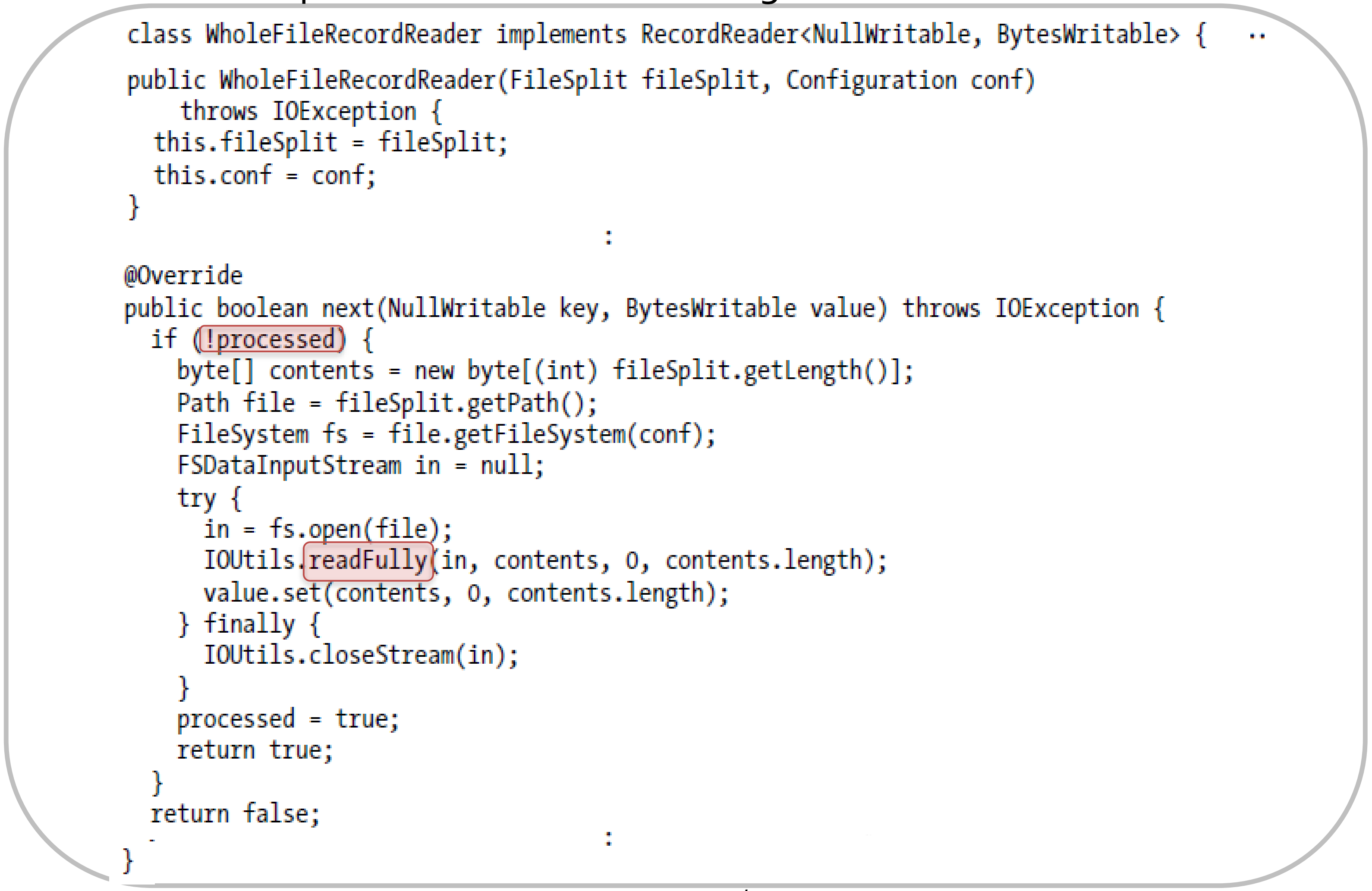
- Some application don't want files to be split
  - Want to process entire data by a single mapper

- Two ways of ensuring an existing file is not split
  - Set the minimum split size larger than the largest file size
  - Override the isSplitable() method to return false

Input Formats

Input Splits and Records – Processing a whole file as a record

- WholeFileRecordReader
  - Take a FileSplit and convert it into a single record

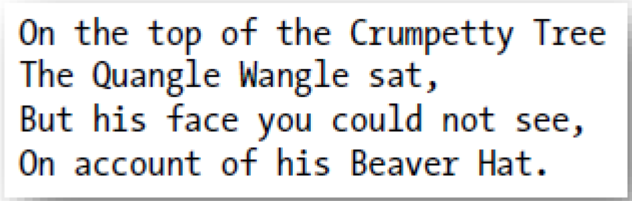


```
class WholeFileRecordReader implements RecordReader<NullWritable, BytesWritable> {   ..

public WholeFileRecordReader(FileSplit fileSplit, Configuration conf)
    throws IOException {
  this.fileSplit = fileSplit;
  this.conf = conf;
}
                                                  :

@Override
public boolean next(NullWritable key, BytesWritable value) throws IOException {
  if (!processed) {
    byte[] contents = new byte[(int) fileSplit.getLength()];
    Path file = fileSplit.getPath();
    FileSystem fs = file.getFileSystem(conf);
    FSDataInputStream in = null;
    try {
      in = fs.open(file);
      IOUtils.readFully(in, contents, 0, contents.length);
      value.set(contents, 0, contents.length);
    } finally {
      IOUtils.closeStream(in);
    }
    processed = true;
    return true;
  }
  return false;
                                                  :
}
```

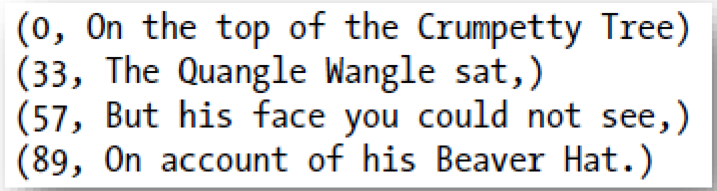
Input Formats

Text Input - TextInputFormat

- TextInputFormat is the default InputFormat
  - Key: The byte offset of the beginning of the line (LongWritable) ; Not line number
  - Value: The contents of the line excluding any line terminators (Text)



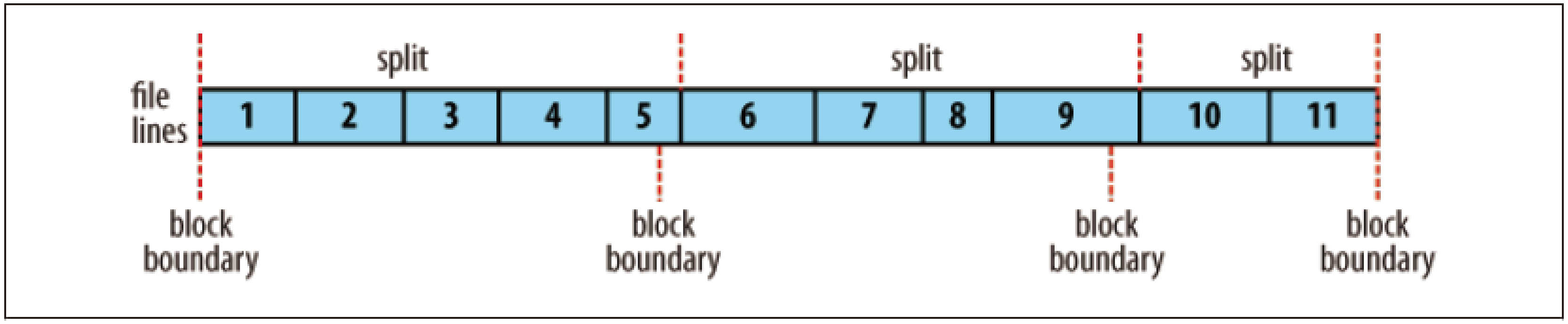
```
On the top of the Crumpetty Tree
The Quangle Wangle sat,
But his face you could not see,
On account of his Beaver Hat.
```



```
(0, On the top of the Crumpetty Tree)
(33, The Quangle Wangle sat,)
(57, But his face you could not see,)
(89, On account of his Beaver Hat.)
```

- Each split knows the size of the preceding splits
  - A global file offset = The offsets within the split + The size of preceding splits

- The logical records do not usually fit into HDFS



Text Input - NLineInputFormat

- Each mapper receives a variable number of lines of input using:
  - **TextInputFormat**, **KeyValueTextInputFormat**
- To receive a fixed number of lines of input, use
  - **NLineInputFormat** as InputFormat
    - N: The number of lines of input
    - Control N in **Mapred.line.input.format.linespermap** property
  - Inefficient if a map task takes a small number of lines of input
    - Due to task setup overhead

Text Input - XML

- Use **StreamXmlRecordReader** class for XML
  - **Org.apache.hadoop.streaming** package
  - Set stream.recordreader.class to org.apache.hadoop.streamin.StreamXmlRecordReader

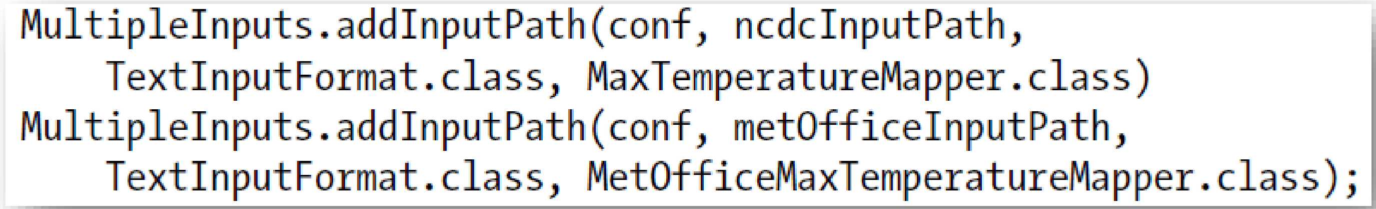
Input Formats

Binary Input

- SequenceFileInputFormat
  - Hadoop’s sequence file format stores sequences of binary key-value pairs
    - Data is splittable (Data has sync points)
    - Use SequenceFileInputFormat
- SequenceFileAsTextInputFormat
  - Convert the sequence file’s keys and values to Text objects
    - Use **toString()** method
- SequenceFileAsBinaryInputFormat
  - Retrieve the sequence file’s keys and values as opaque binary objects

Multiple Inputs

- Use MultipleInput when
  - Have data sources that provide the same type of data but in different formats
    - Need to be parsed differently
    - Ex) One might be tab-separated plain text, the other a binary sequence file



```
MultipleInputs.addInputPath(conf, ncdcInputPath,
    TextInputFormat.class, MaxTemperatureMapper.class)
MultipleInputs.addInputPath(conf, metOfficeInputPath,
    TextInputFormat.class, MetOfficeMaxTemperatureMapper.class);
```

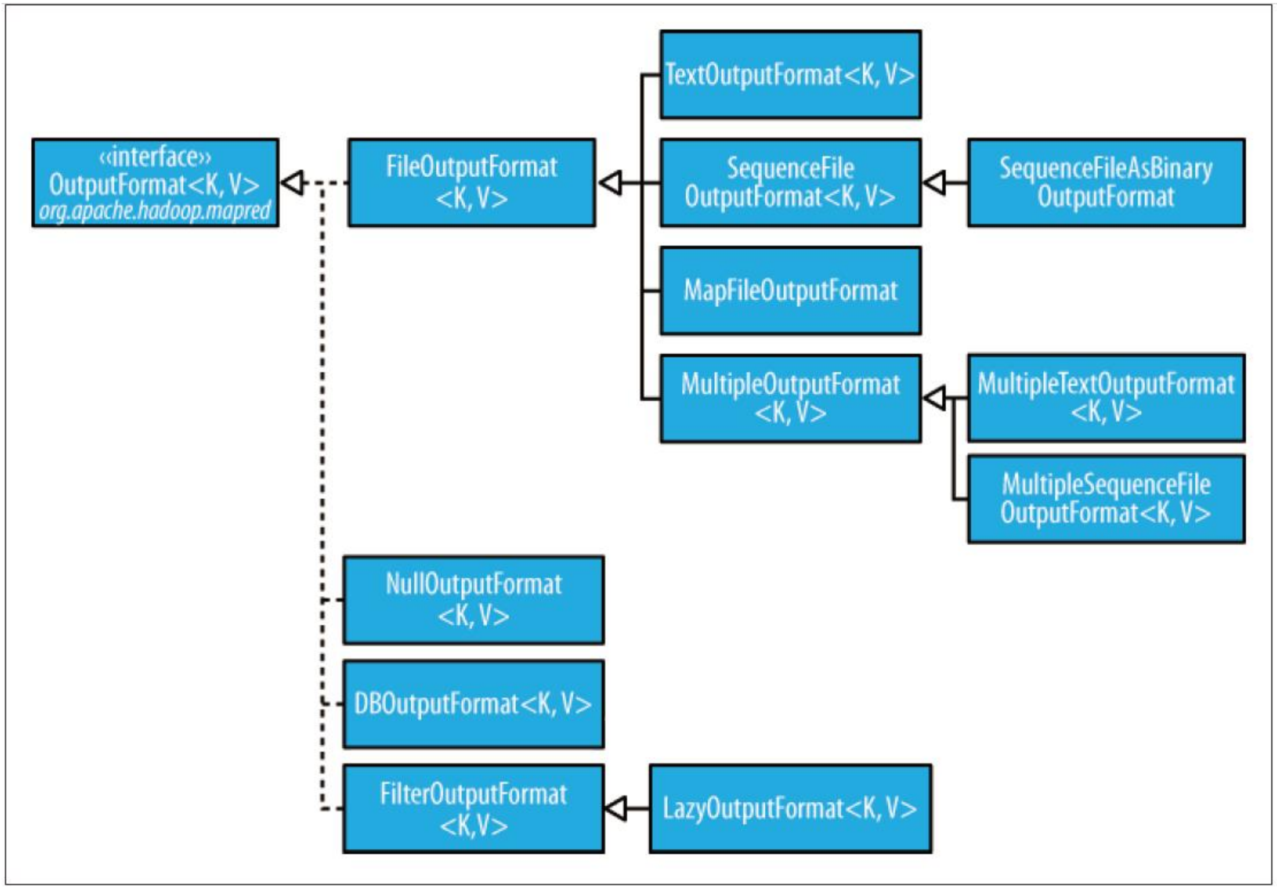
  - Use different mappers
  - The map outputs have the same types
    - Reducers are not aware of the different mappers

Outline

- MapReduce Types
- Input Formats
- Output Formats
  - Text Output
  - Binary Output
  - Multiple Outputs
  - Lazy Output
  - Database Output

Output Formats

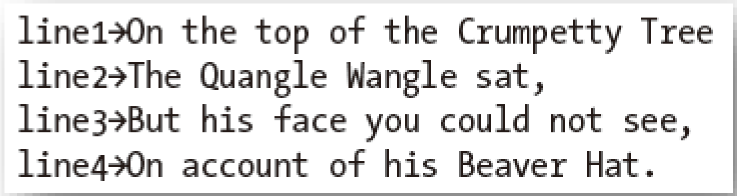
- The OutputFormat class hierarchy


«interface» OutputFormat<K, V> org.apache.hadoop.mapred
FileOutputFormat <K, V>
TextOutputFormat<K, V>
SequenceFile OutputFormat<K, V>
SequenceFileAsBinary OutputFormat
MapFileOutputFormat
MultipleOutputFormat <K, V>
MultipleTextOutputFormat <K, V>
MultipleSequenceFile OutputFormat<K, V>
NullOutputFormat <K, V>
DBOutputFormat<K, V>
FilterOutputFormat <K,V>
LazyOutputFormat<K, V>

Output Formats

Text Output

- TextOutputFormat (default)
  - Write records as lines of text
  - Keys and Values may be of any type
    - It calls toString() method
  - Each key-value pair is separated by a tab character
    - Set the separator in **mapred.textoutputformat.separator** property



```
line1→On the top of the Crumpetty Tree
line2→The Quangle Wangle sat,
line3→But his face you could not see,
line4→On account of his Beaver Hat.
```

Binary Output

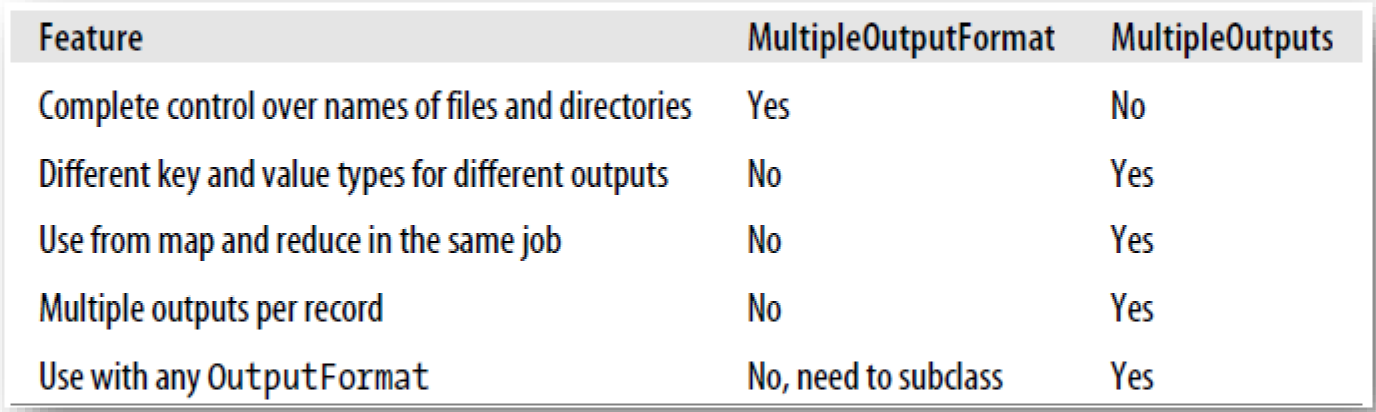
- SequenceFileOutputFormat
  - Write sequence files for its output
  - Compact, readily compressed (Useful for a further MapReduce job)
- SequenceFileAsBinaryOutputFormat
  - Write keys and values in raw binary format into a SequenceFile container
- MapFileOutputFormat
  - Write MapFiles as output
  - The keys in MapFile must be added in order
    - Ensure that the reducers emit keys in sorted order (only for this format)

Multiple Output

- MultipleOutputFormat and MultipleOutputs
  - Help to produce multiple files per reducer
- MultipleOutputFormat
  - The names of multiple files are derived from the output keys and values
  - Is an abstract class with
    - MultipleTextOutputFormat
    - MultipleSequenceFileOutputFormat
- MultipleOutputs
  - Can emit different types for each output (Differ from MultipleOutputFormat)
  - Less control over the naming of outputs

Multiple Output

- Difference between MultipleOutputFormat and MultiplOutputs



Feature	MultipleOutputFormat	MultipleOutputs
Complete control over names of files and directories	Yes	No
Different key and value types for different outputs	No	Yes
Use from map and reduce in the same job	No	Yes
Multiple outputs per record	No	Yes
Use with any `OutputFormat`	No, need to subclass	Yes

  - MultipleOutputs is more fully featured
  - MultipleOutputFormat has more control over the output directory structure and file naming

Output Formats

Lazy Output

- LazyOutput helps some applications that doesn’t want to create empty files
  - Since FileOutputFormat subclasses create output files even if they are empty
- To use it
  - Call its **setOutputFormatClass()** method with the JobConf option