Overview

Know the important class hierarchies in the Java MapReduce framework

Write and execute your first Mapper, Reducer and Driver classes

Demo

Download Hadoop jars

Setup a MapReduce project in IntelliJ

Implementing a MapReduce Job

Hadoop is an open source framework for running MapReduce jobs

Since Hadoop is written using Java, MapReduce jobs are usually written in Java as well

Implementing a MapReduce Job

A class where the map logic is implemented

A class where the reduce logic is implemented

A driver program that sets up the job

Implementing a MapReduce Job

Map

A class where the map logic is implemented

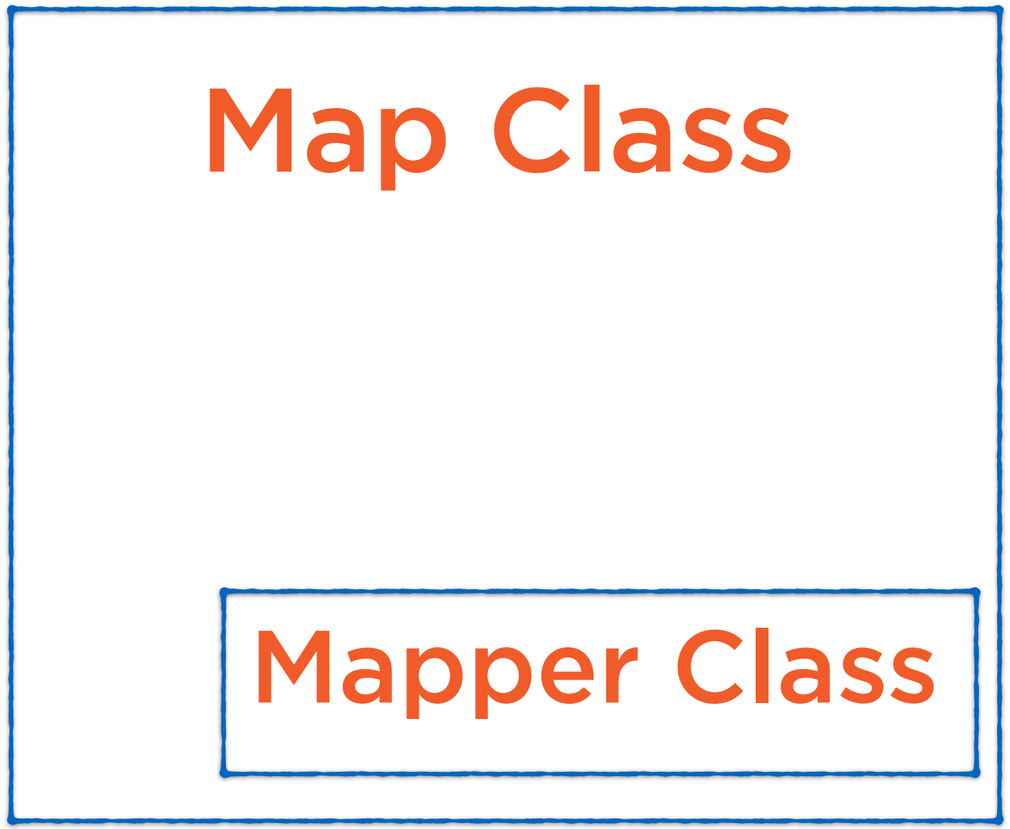
Reduce

A class where the reduce logic is implemented

Main

A driver program that sets up the job

The Map Step



The map logic is implemented in a class that extends the Mapper Class

The Map Step

Map Class

```
<input key type,
input value type,
output key type,
output value type>
```

Mapper Class

This is a generic class, with 4 type parameters

The Map Step

input key type,
input value type

<K,V>

Map

The input key, value pair is determined by how the input file on disk is read

The Map Step

input key type,
input value type

<K,V>

Map

Input Text File

first line in a file
second line in a file
third line in a file

The Map Step

input key type,
input value type

<K,V>

Map

Input Key Value pairs

<1, first line in a file>
<2, second line in a file>
<3, third line in a file>

The Map Step

Map Class

```
<input key type,
input value type,
output key type,
output value type>
```

Mapper Class

This is a generic class, with 4 type parameters

The Map Step

output key type,
output value type

<K,V>

Map

The output key, value pair depends on the processing you do in the map phase

The Map Step

output key type,
output value type

<K,V>

Map

Input Key Value pairs

<1, first line in a file>
<2, second line in a file>
<3, third line in a file>

The Map Step

output key type,
output value type

<K,V>

Map

Output Key Value pairs

<first, 1>
<line, 1>
<in, 1>
<a, 1>
<file, 1>

Implementing a MapReduce Job

Map

A class where the map logic is implemented

Reduce

A class where the reduce logic is implemented

Main

A driver program that sets up the job

Implementing a MapReduce Job

A class where the map logic is implemented

A class where the reduce logic is implemented

A driver program that sets up the job

Implementing a MapReduce Job

Map

A class where the map logic is implemented

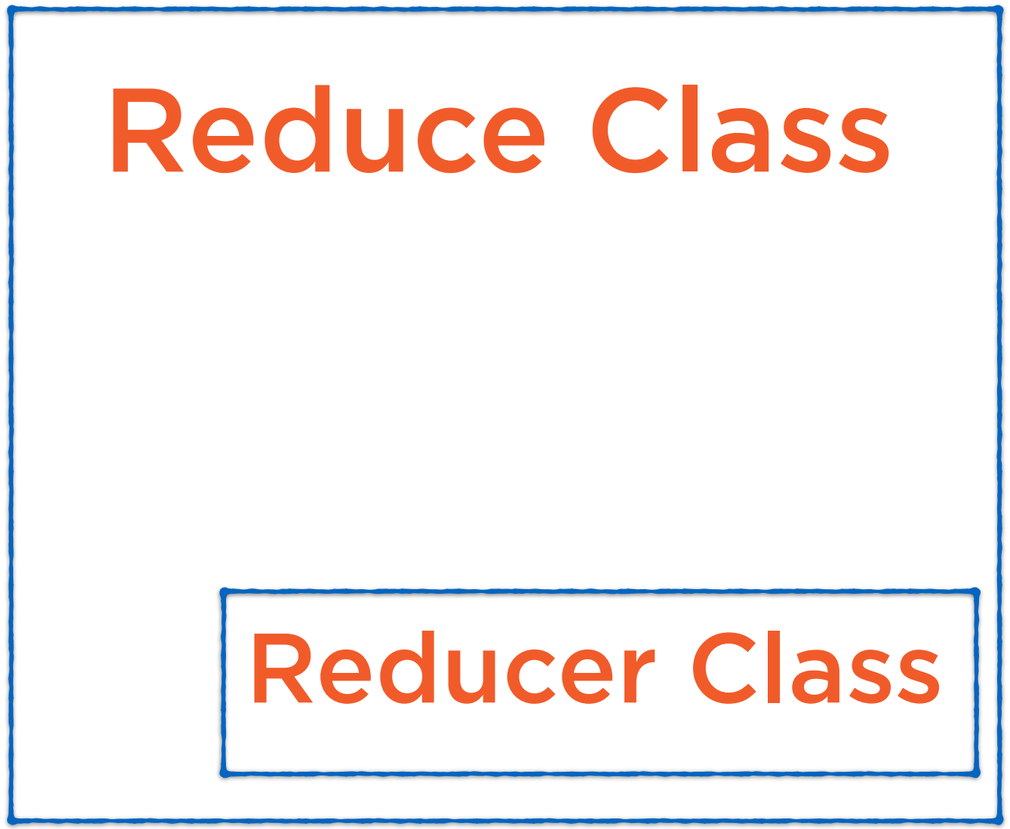
Reduce

A class where the reduce logic is implemented

Main

A driver program that sets up the job

The Reduce Step



The reduce logic is implemented in a class that extends the Reducer Class

The Reduce Step

Reduce Class

```
<input key type,
input value type,
output key type,
output value type>
```

Reducer Class

This is also a generic class, with 4 type parameters

The Reduce Step

input key type,
input value type

<K,V>

Reduce

The input key, value pair is the same as the output of the map phase

input key type,
input value type

<K,V>

Reduce

The Reduce Step

Map Output Key Value pairs

= Reduce Input Key Value pairs

<first, 1>
<line, 1>
<in, 1>
<a, 1>
<file, 1>

The Reduce Step

Reduce Class

```
<input key type,
input value type,
output key type,
output value type>
```

Reducer Class

This is also a generic class, with 4 type parameters

The Reduce Step

output key type,
output value type

<K,V>

Reduce

The output key, value pair depends on the processing you do in the reduce phase

The Reduce Step

output key type,
output value type

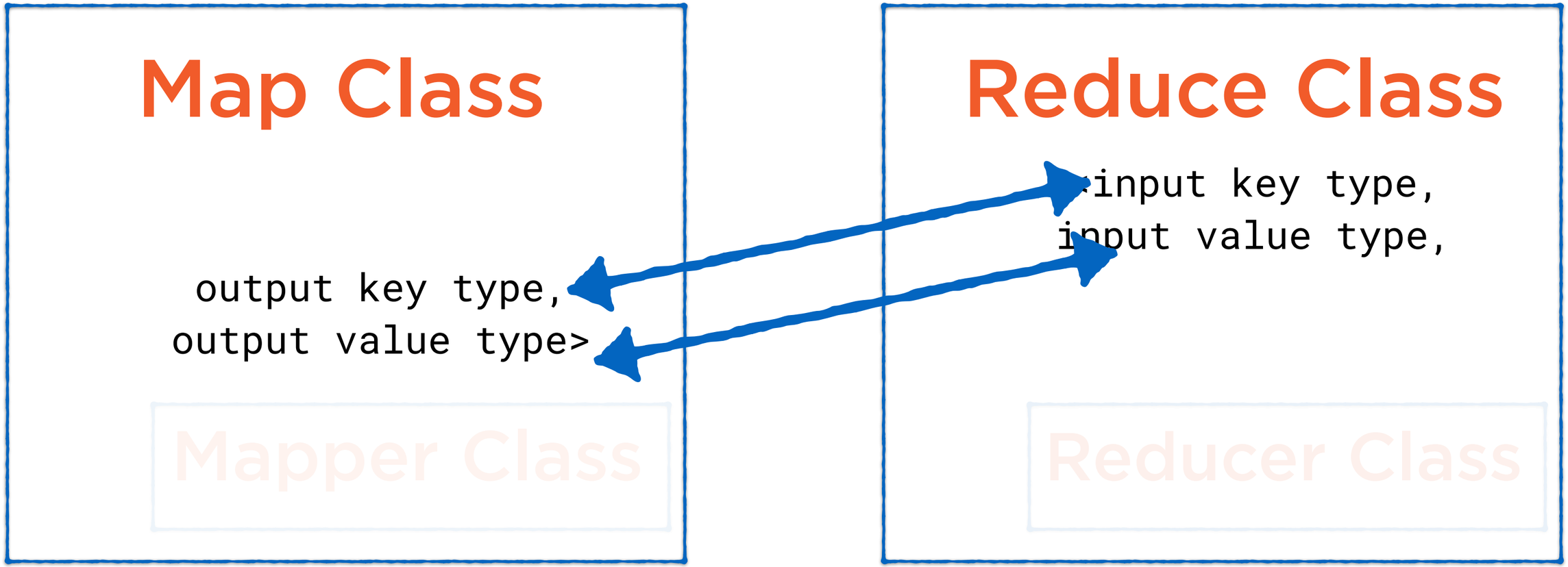
<K,V>

Reduce

Reduce Output Key Value pairs

<first, 1>
<second, 1>
<third, 1>
<line, 3>
<in, 3>
<a, 3>
<file, 3>

Matching Data Types



The output types of the Mapper should match the input types of the Reducer

Implementing a MapReduce Job

Map

A class where the map logic is implemented

Reduce

A class where the reduce logic is implemented

Main

A driver program that sets up the job

Implementing a MapReduce Job

A class where the map logic is implemented

A class where the reduce logic is implemented

A driver program that sets up the job

Implementing a MapReduce Job

Map

A class where the map logic is implemented

Reduce

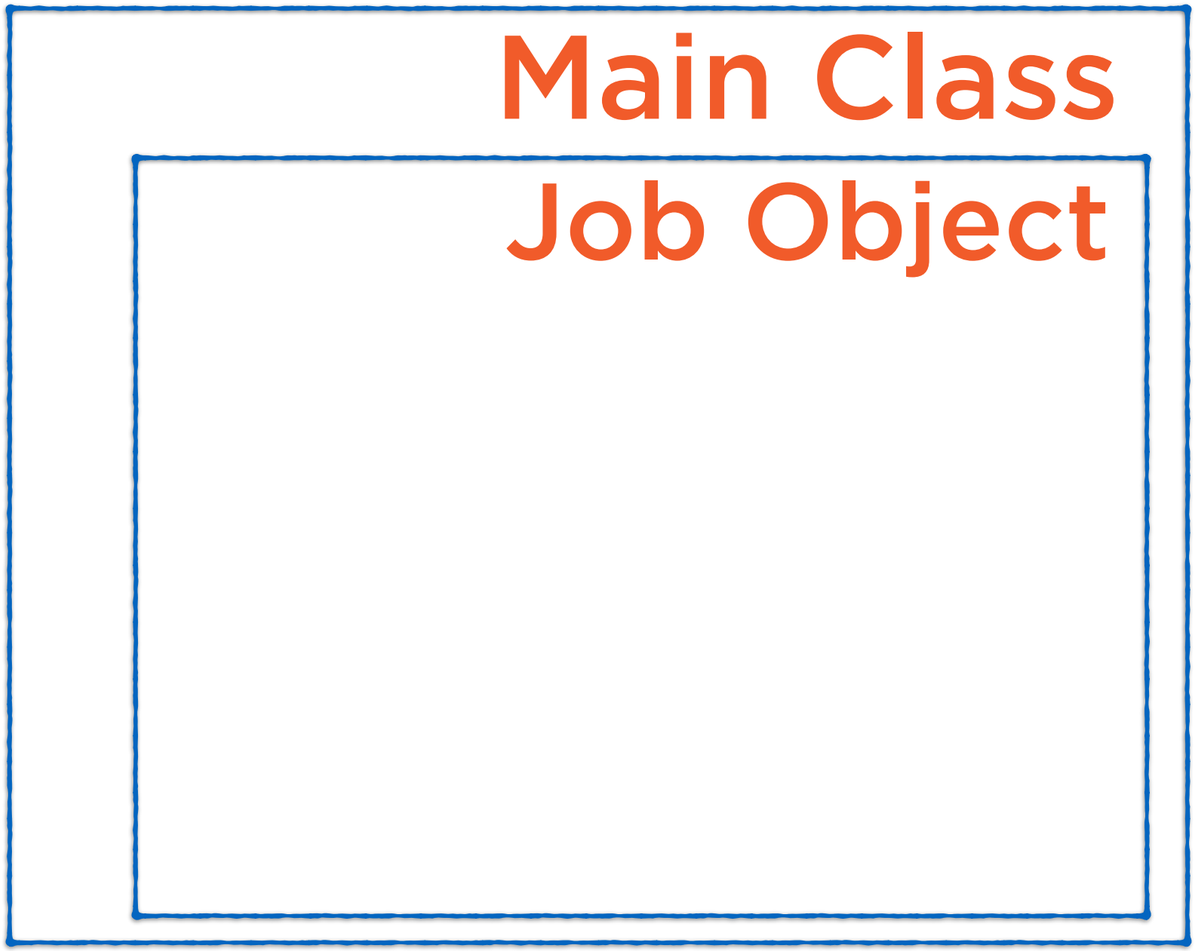
A class where the reduce logic is implemented

Main

A driver program that sets up the job

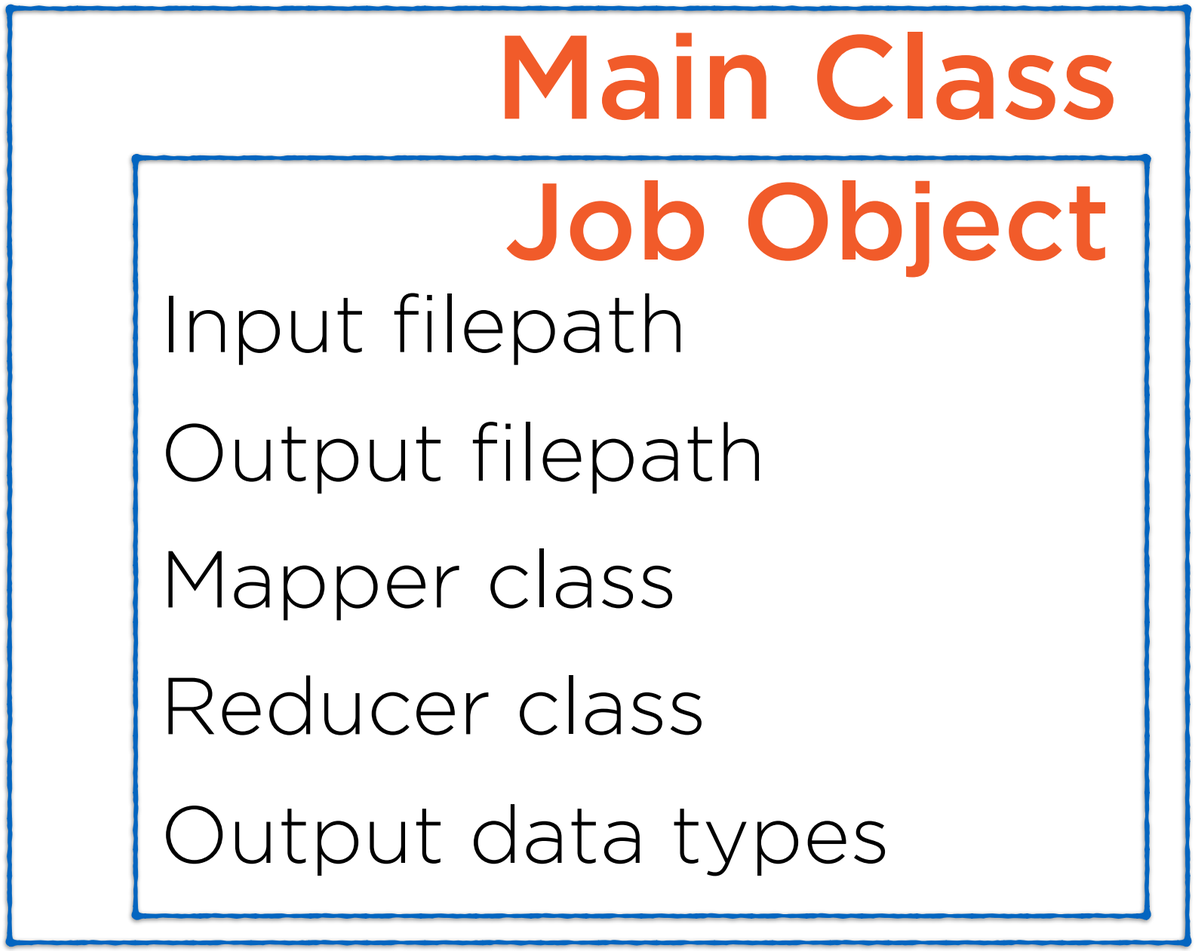
Setting up the Job

The Mapper and Reducer classes are used by a Job that is configured in the Main Class



Setting up the Job

The Job has a bunch of properties that need to be configured



Setting up the Job

Main

Input File Path

The location of the input data files which is fed to the MapReduce

Setting up the Job

Main

Output File Path

A non-existent directory where the resultant files will be written

Setting up the Job

Mapper and Reducer Class

Main

The Java class where we've written the code for our implementation

Setting up the Job

Main

Output data types

The data types of the final result - again in the form of key, value pairs

Implementing a MapReduce Job

Map

A class where the map logic is implemented

Reduce

A class where the reduce logic is implemented

Main

A driver program that sets up the job

Implementing a MapReduce Job

A class where the map logic is implemented

A class where the reduce logic is implemented

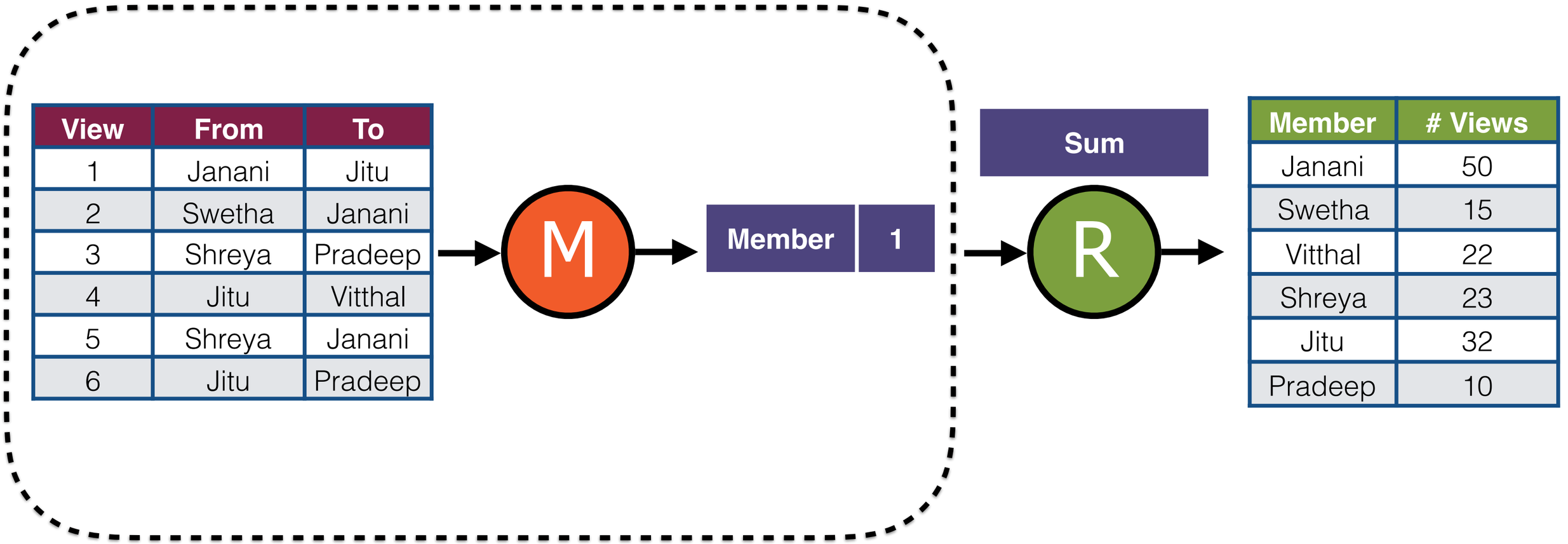
A driver program that sets up the job

Demo

Set up the input file path

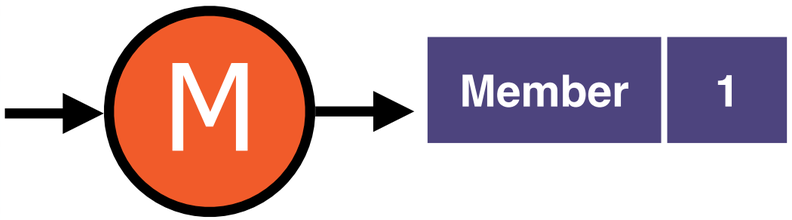
Set up a simple rawViews.txt file

Building a User-ViewCount Map



The Map Step

View	From	To
1	Janani	Jitu
2	Swetha	Janani
3	Shreya	Pradeep
4	Jitu	Vitthal
5	Shreya	Janani
6	Jitu	Pradeep



This logic is implemented in a class that extends Mapper

The Map Step

View	From	To
1	Janani	Jitu
2	Swetha	Janani
3	Shreya	Pradeep
4	Jitu	Vitthal
5	Shreya	Janani
6	Jitu	Pradeep

The Mapper will read 1 line at a time from this dataset

The Map Step

View	From	To
1	Janani	Jitu
2	Swetha	Janani
3	Shreya	Pradeep
4	Jitu	Vitthal
5	Shreya	Janani
6	Jitu	Pradeep

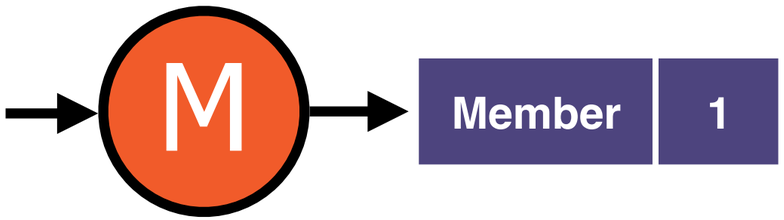
Input

{LineNum, Line}

{Long, String}

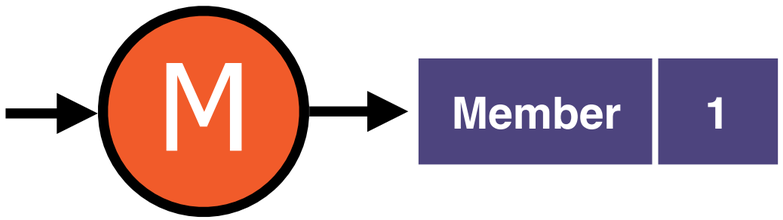
Datatypes

The Map Step



It will parse the member name from the line and output a {key,value} pair

The Map Step



Datatypes

{String, Integer}

A Mapper Class

```
public class Map extends
Mapper<LongWritable,Text,Text,IntWritable> {
@Override

public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException{
    String[] row = value.toString().split("\t");
    context.write(new Text(row[2]),new IntWritable(1));
}}
```

```
public class Map extends
Mapper<LongWritable,Text,Text,IntWritable> {
…
}
```

Input Key Type

The Line number in the data set being read

A Hadoop Writable class which wraps around Java Long

```
public class Map extends
Mapper<LongWritable,Text,Text,IntWritable> {
…
}
```

Input Value Type

The actual line in the data set being read

A Hadoop Writable class which wraps around Java String

```
public class Map extends
Mapper<LongWritable,Text,Text,IntWritable> {
…
}
```

Output Key Type

The member name in the current line

A Hadoop Writable class which wraps around Java String

```
public class Map extends
Mapper<LongWritable,Text,Text,IntWritable> {
…
}
```

Output Value Type

The view count for the member from this line

A Hadoop Writable class which wraps around Java Integer

A Mapper Class

```
public class Map extends
Mapper<LongWritable,Text,Text,IntWritable> {
@Override

public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException{
    String[] row = value.toString().split("\t");
    context.write(new Text(row[2]),new IntWritable(1));
}}
```

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

String[] row =
value.toString().split("\t");


context.write(new Text(row[2]),new
IntWritable(1));
}
```

The Map logic is implemented by overriding the map method

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

String[] row =
value.toString().split("\t");



context.write(new Text(row[2]),new
IntWritable(1));
}
```

The map method arguments are key and value and a **context**

The context stores the output key, value pairs

The context is where the internal Shuffle, Sort logic will happen before passing data to the Reducer Class

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

String[] row =
value.toString().split("\t");


context.write(new Text(row[2]),new
IntWritable(1));
}
```

- Take the line and split it into words
- The third word is the Member name

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

String[] row =
value.toString().split("\t");


context.write(new Text(row[2]),new
IntWritable(1));
}
```

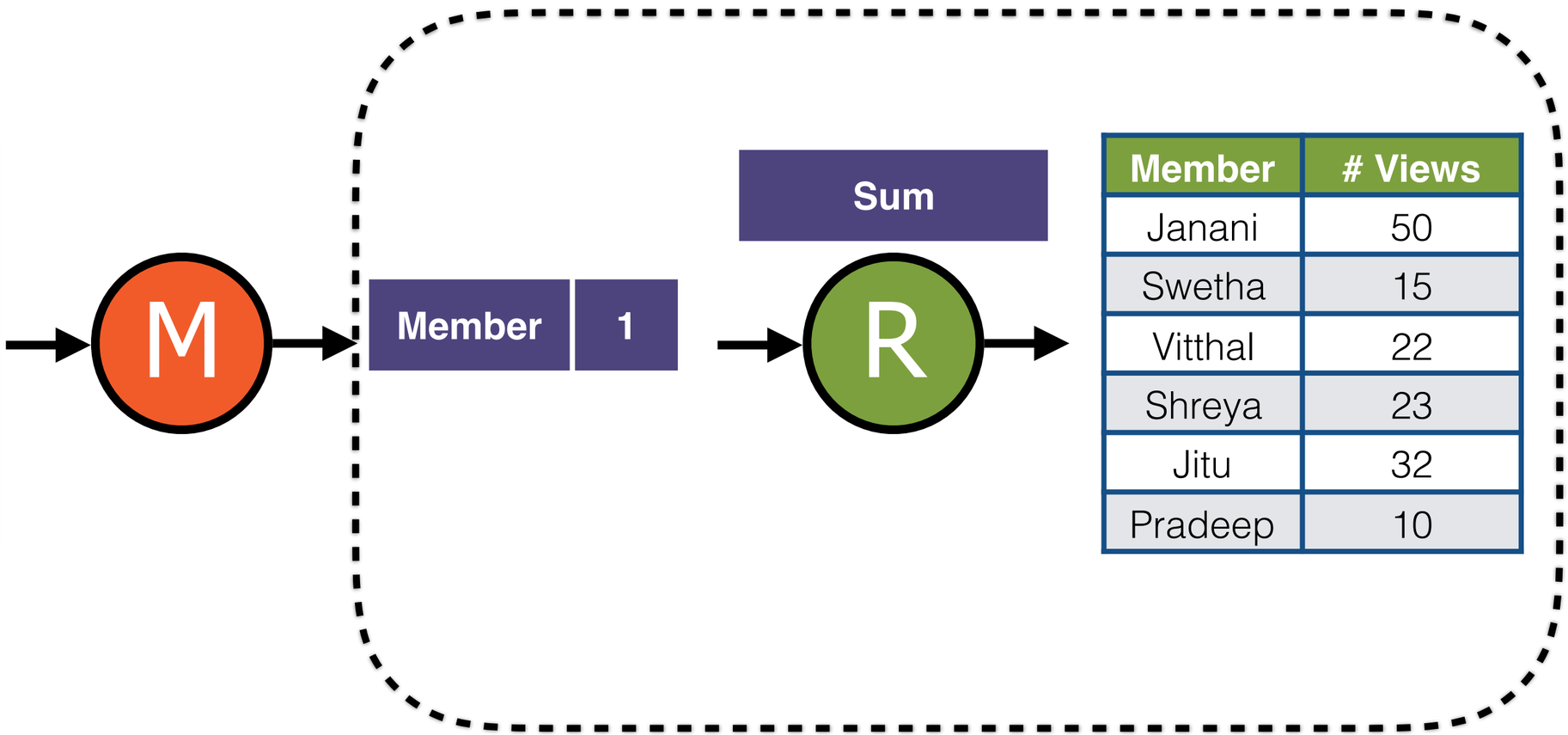
- The output value of the map is always the integer 1
- Each line corresponds to 1 view of the profile

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

String[] row =
value.toString().split("\t");


context.write(new Text(row[2]),new
IntWritable(1));
}
```

- Write the key value pair with appropriate data types to the context

Building a User-ViewCount Map

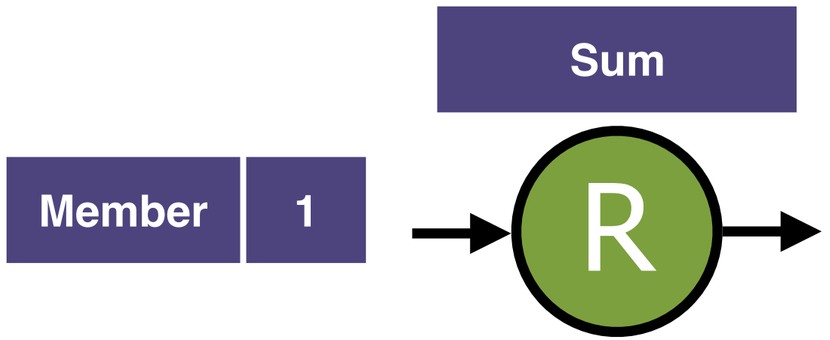
View	From	To
1	Janani	Jitu
2	Swetha	Janani
3	Shreya	Pradeep
4	Jitu	Vitthal
5	Shreya	Janani
6	Jitu	Pradeep



Member	# Views
Janani	50
Swetha	15
Vitthal	22
Shreya	23
Jitu	32
Pradeep	10

The Reduce Step

This logic is implemented in a class that extends Reducer

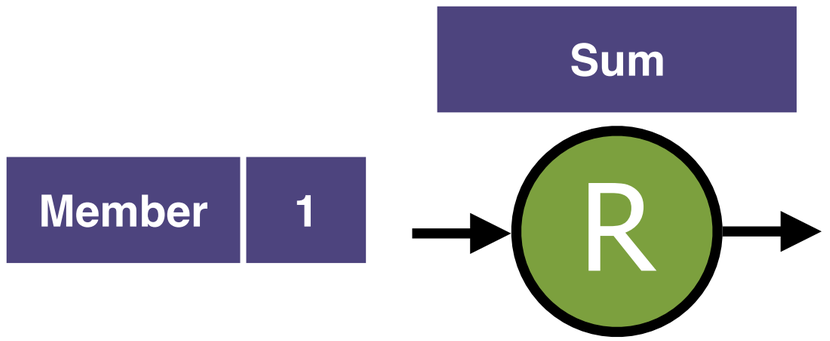


Member	# Views
Janani	50
Swetha	15
Vitthal	22
Shreya	23
Jitu	32
Pradeep	10

The Reduce Step

For each member, the Reducer will get a collection

The collection holds all the corresponding values for the same key



Member	# Views
Janani	50
Swetha	15
Vitthal	22
Shreya	23
Jitu	32
Pradeep	10

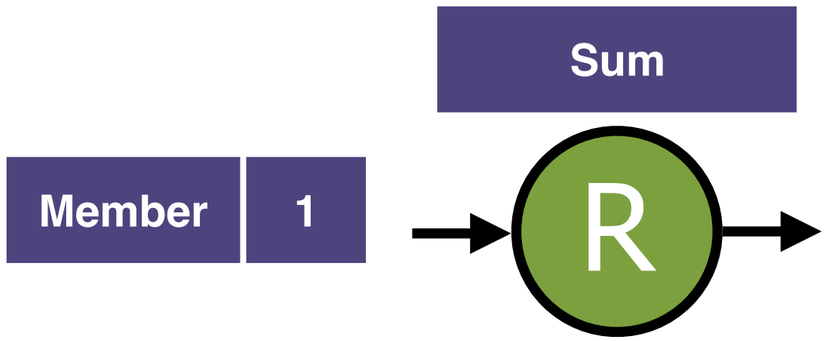
The Reduce Step

Input

{Member,
Collection of counts}

{String,
Iterable<Integer>}

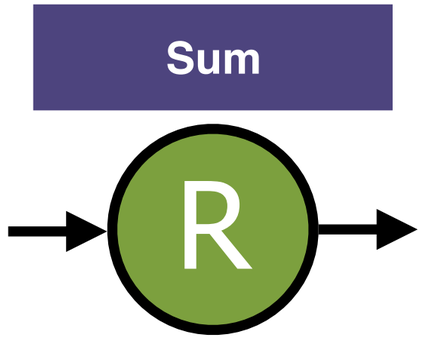
Datatypes



Member	# Views
Janani	50
Swetha	15
Vitthal	22
Shreya	23
Jitu	32
Pradeep	10

The Reduce Step

It will sum up all the view counts for a given member



Member	# Views
Janani	50
Swetha	15
Vitthal	22
Shreya	23
Jitu	32
Pradeep	10

The Reduce Step

Datatypes

{String, Integer}

Member	# Views
Janani	50
Swetha	15
Vitthal	22
Shreya	23
Jitu	32
Pradeep	10

A Reducer Class

```
public class Reduce extends Reducer<Text,
IntWritable,Text,IntWritable> {
    @Override
    public void reduce(Text key, Iterable<IntWritable> values,
     Context context) throws IOException, InterruptedException{
        int count = 0;
        for (IntWritable value:values)
        {
            count += value.get();
        }
        context.write(key, new IntWritable(count));
}}
```

```
public class Reduce extends
Reducer<Text,IntWritable,Text,IntWritable> {…
}
```

Input Key Type

The member name

Should match the Output Key Type of the Mapper

```
public class Reduce extends
Reducer<Text,IntWritable,Text,IntWritable> {…
}
```

Input Value Type

Input count (1) for this member

The reducer will see an Iterable with all the counts collected for 1 member

The type should match the Output value type of the mapper

```
public class Reduce extends
Reducer<Text,IntWritable,Text,IntWritable> {…
}
```

Output Key Type

The member name

```
public class Reduce extends
Reducer<Text,IntWritable,Text,IntWritable> {…
}
```

Output Value Type

The total view count for this member

A Reducer Class

```
public class Reduce extends Reducer<Text,
IntWritable,Text,IntWritable> {
    @Override
    public void reduce(Text key, Iterable<IntWritable> values,
     Context context) throws IOException, InterruptedException{
        int count = 0;
        for (IntWritable value:values)
        {
            count += value.get();
        }
        context.write(key, new IntWritable(count));
}}
```

```java
@Override
public void reduce(Text key,
Iterable<IntWritable> values,
Context context)
throws IOException,
InterruptedException{

int count = 0;
for (IntWritable value:values {
 count += value.get();
}

context.write(key, new
IntWritable(count));
}
```

The Reduce logic is implemented by overriding the reduce method

```
@Override
public void reduce(Text key,
Iterable<IntWritable> values,
Context context)
throws IOException,
InterruptedException{

int count = 0;
for (IntWritable value:values {
 count += value.get();
}

context.write(key, new
IntWritable(count));
}
```

The reduce method arguments are input key, **iterable** of values and a context

The context is an object that stores the output key, value pairs

```
@Override
public void reduce(Text key,
Iterable<IntWritable> values,
Context context)
throws IOException,
InterruptedException{

int count = 0;
for (IntWritable value:values {
 count += value.get();
}

context.write(key, new
IntWritable(count));
}
```

For this key

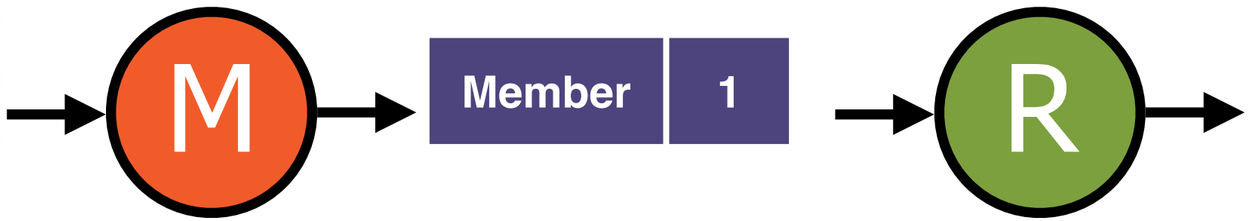
- Initialize a count to zero
- Iterate through the values and compute the sum

```
@Override
public void reduce(Text key,
Iterable<IntWritable> values,
Context context)
throws IOException,
InterruptedException{

int count = 0;
for (IntWritable value:values {
 count += value.get();
}

context.write(key, new
IntWritable(count));
}
```

- Write the key value pair with appropriate data types to the context

Building a User-ViewCount Map

View	From	To
1	Janani	Jitu
2	Swetha	Janani
3	Shreya	Pradeep
4	Jitu	Vitthal
5	Shreya	Janani
6	Jitu	Pradeep



Member	# Views
Janani	50
Swetha	15
Vitthal	22
Shreya	23
Jitu	32
Pradeep	10

We need to set up a Job to execute this

The Job

```
public class ViewCount extends Configured implements Tool {
    @Override
    public int run(String[] args) throws Exception{…}

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

```
public class ViewCount extends Configured implements Tool {
@Override
public int run(String[] args) throws Exception{…}

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

Boiler Plate to Set Up the Job

The Configured class allows us to use a Configuration object to specify parameters

Preferable to hardcoding configuration values within the class itself

```
public class ViewCount extends Configured implements Tool {
@Override
public int run(String[] args) throws Exception{…}

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

Boiler Plate to Set Up the Job

The Tool interface supports handling of command line options

Used along with the ToolRunner class which picks up those options specific to the application that we are currently running

```
public class ViewCount extends Configured implements Tool {
@Override
public int run(String[] args) throws Exception{…}

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

Boiler Plate to Set Up the Job

This is where the Job object is instantiated and configured

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

Boiler Plate *main* Method

The ToolRunner class extracts the right command line options which this MapReduce needs

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

Boiler Plate *main* Method

Instantiate the Main Class

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

Boiler Plate *main* Method

Any command line arguments to run this code

The input directory with the data file

The output directory where the result data is to be stored

The Job

```
public class ViewCount extends Configured implements Tool {
    @Override
    public int run(String[] args) throws Exception{…}

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

```java
@Override
public int run(String[] args) throws Exception{

    Configuration conf = this.getConf();

    Job job = Job.getInstance(conf);
    job.setJobName("viewCount");
    job.setJarByClass(ViewCount.class);

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

    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

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

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

The run method where the Job is instantiated

```java
@Override
public int run(String[] args) throws Exception{

    Configuration conf = this.getConf();

    Job job = Job.getInstance(conf);
    job.setJobName("viewCount");
    job.setJarByClass(ViewCount.class);

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

    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

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

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

- Get the configuration object from the base class

```
@Override
public int run(String[] args) throws Exception{

    Configuration conf = this.getConf();

Job job = Job.getInstance(conf);
job.setJobName("viewCount");
job.setJarByClass(ViewCount.class);

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

    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

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

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

- Get a Job Instance
- Set the Job name
- Set the class to the driver program

```
@Override
public int run(String[] args) throws Exception{

    Configuration conf = this.getConf();

    Job job = Job.getInstance(conf);
    job.setJobName("viewCount");
    job.setJarByClass(ViewCount.class);

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

job.setMapperClass(Map.class);
job.setReducerClass(Reduce.class);

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

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

- Set the output data types of the reducer
- Set the Mapper Class and Reducer Class

```
@Override
public int run(String[] args) throws Exception{

    Configuration conf = this.getConf();

    Job job = Job.getInstance(conf);
    job.setJobName("viewCount");
    job.setJarByClass(ViewCount.class);

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

    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

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

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

- Point to the input file path
- Point to the output file path

- The Job reads from file and writes to file

```java
@Override
public int run(String[] args) throws Exception{

    Configuration conf = this.getConf();

    Job job = Job.getInstance(conf);
    job.setJobName("viewCount");
    job.setJarByClass(ViewCount.class);

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

    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    Path inputFilePath = new Path(args[0]);
    Path outputFilePath = new Path(args[1]);
    FileInputFormat.addInputPath(job, inputFilePath);
    FileOutputFormat.setOutputPath(job, outputFilePath);
return job.waitForCompletion(true) ? 0 : 1;
}
```

◀ Start the job

Demo

Set up the input arguments

Run the MapReduce code on our local machine

Summary

Understood how to use the MapReduce framework to set up and run your very first MapReduce

Understood basic framework details used to configure your job