K-MEANS CLUSTERING

Here are all the classes we'll need

VectorWritable, ClusterCenter

Both of these are custom WritableComparable wrappers for a DoubleVector class

K-MEANS CLUSTERING

Here are all the classes we'll need

VectorWritable, ClusterCenter

A DoubleVector is a set of doubles that represent the co-ordinates of 1 data point

K-MEANS CLUSTERING

Here are all the classes we'll need

VectorWritable, ClusterCenter

DistanceMeasurer

ManhattanDistance

A way to measure the distance between a data point and a cluster center

K-MEANS CLUSTERING

Here are all the classes we'll need

VectorWritable, ClusterCenter

DistanceMeasurer , ManhattanDistance

KMeansMapper

KMeansReducer

KMeansClusteringJob

K-MEANS CLUSTERING

Here are all the classes we'll need

VectorWritable, ClusterCenter

DistanceMeasurer , ManhattanDistance

KMeansMapper

KMeansReducer

KMeansClusteringJob

The code for all of this is courtesy of a nice blog called codingwiththomas

K-MEANS CLUSTERING

Let's now get into the details

K-MEANS CLUSTERING

Each data point is represented using a set of numbers

p1	3	2.5
p2	1	4
p3	2	3
p4	4	5.5
p5	6	6

K-MEANS CLUSTERING

The operation we'll be doing over and over with these arrays is the measurement of distance

double[ ]

p1	3	2.5
p2	1	4
p3	2	3
p4	4	5.5
p5	6	6

K-MEANS CLUSTERING

There are many possible **Distance Metrics** we could choose from

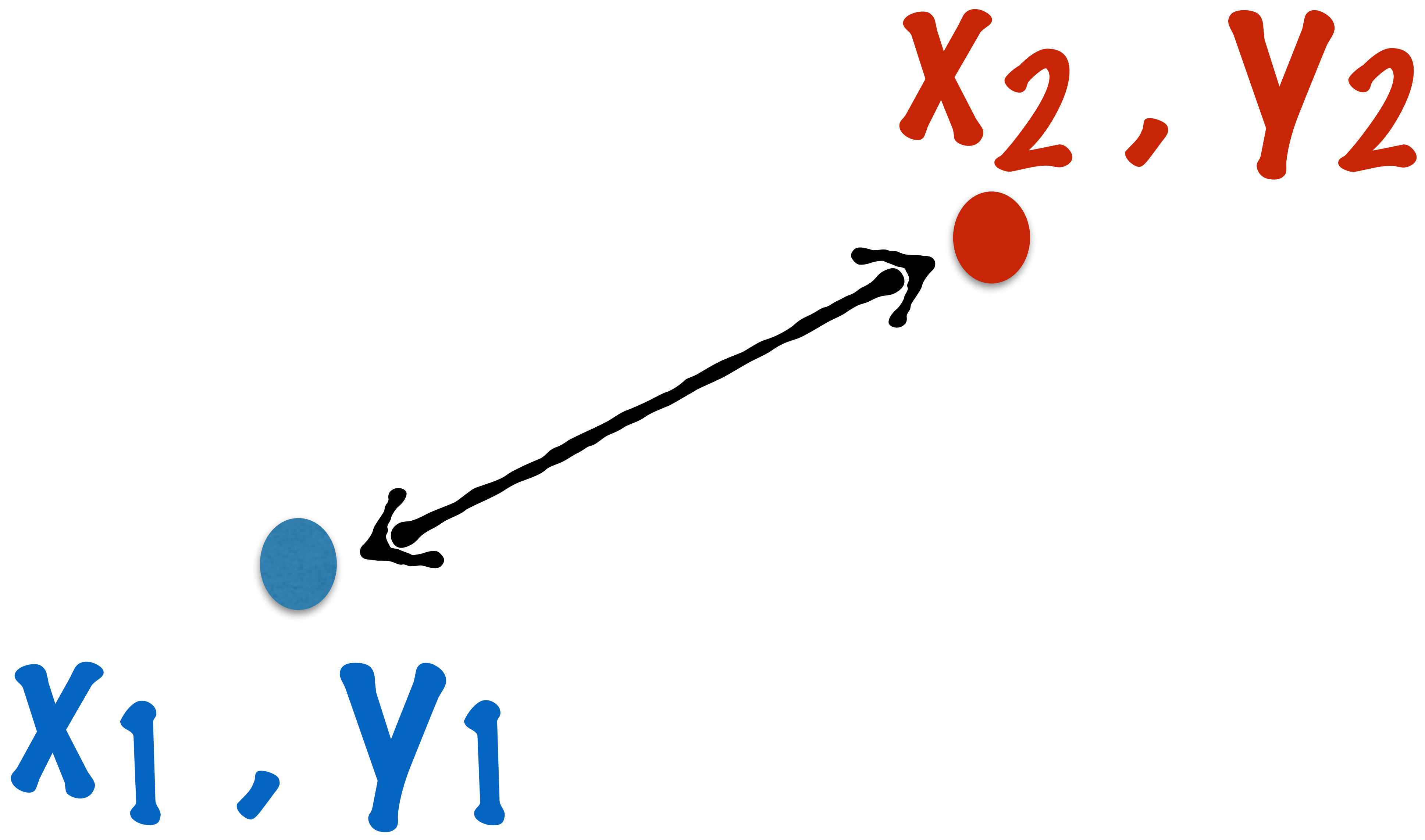
`double[ ]`

p1	3	2.5
p2	1	4
p3	2	3
p4	4	5.5
p5	6	6

K-MEANS CLUSTERING

Distance Metrics

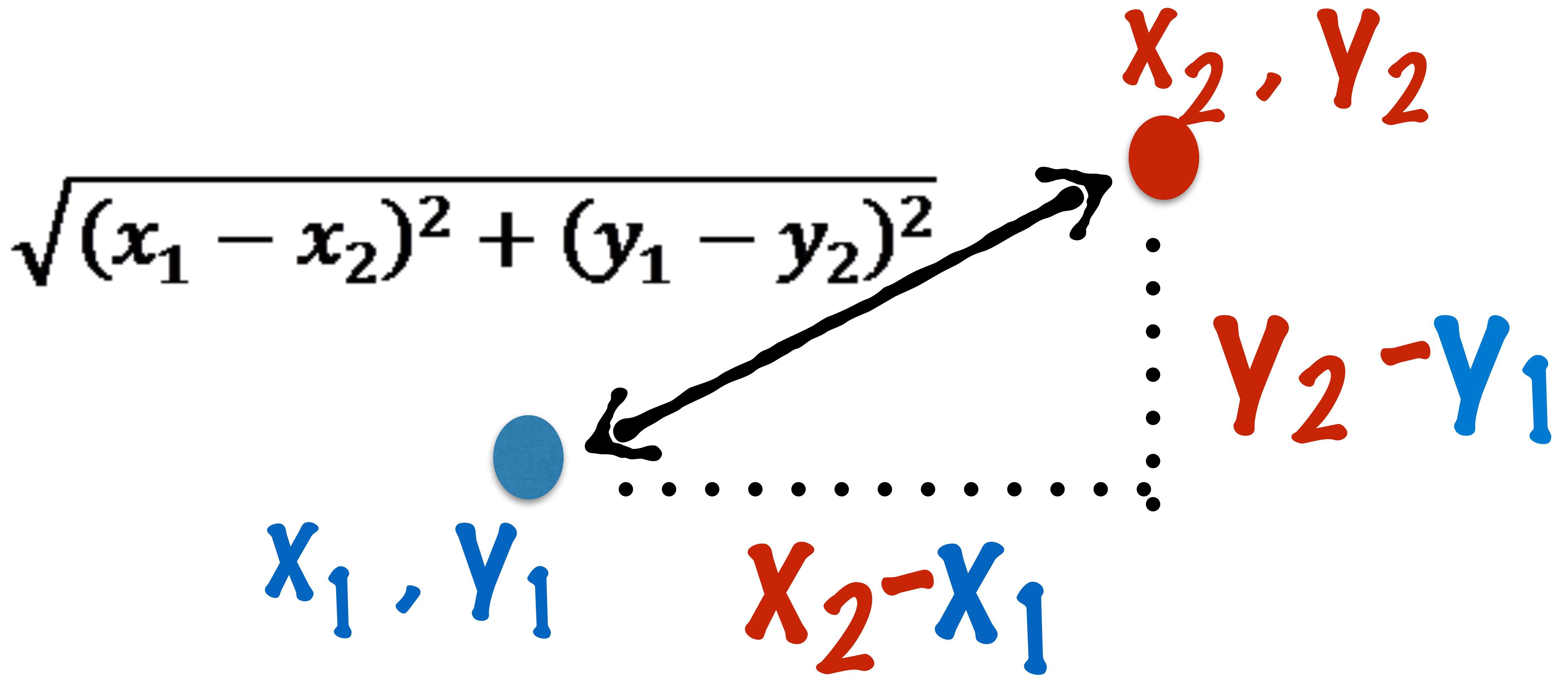
One simple example is the Euclidean distance



K-MEANS CLUSTERING

Distance Metrics

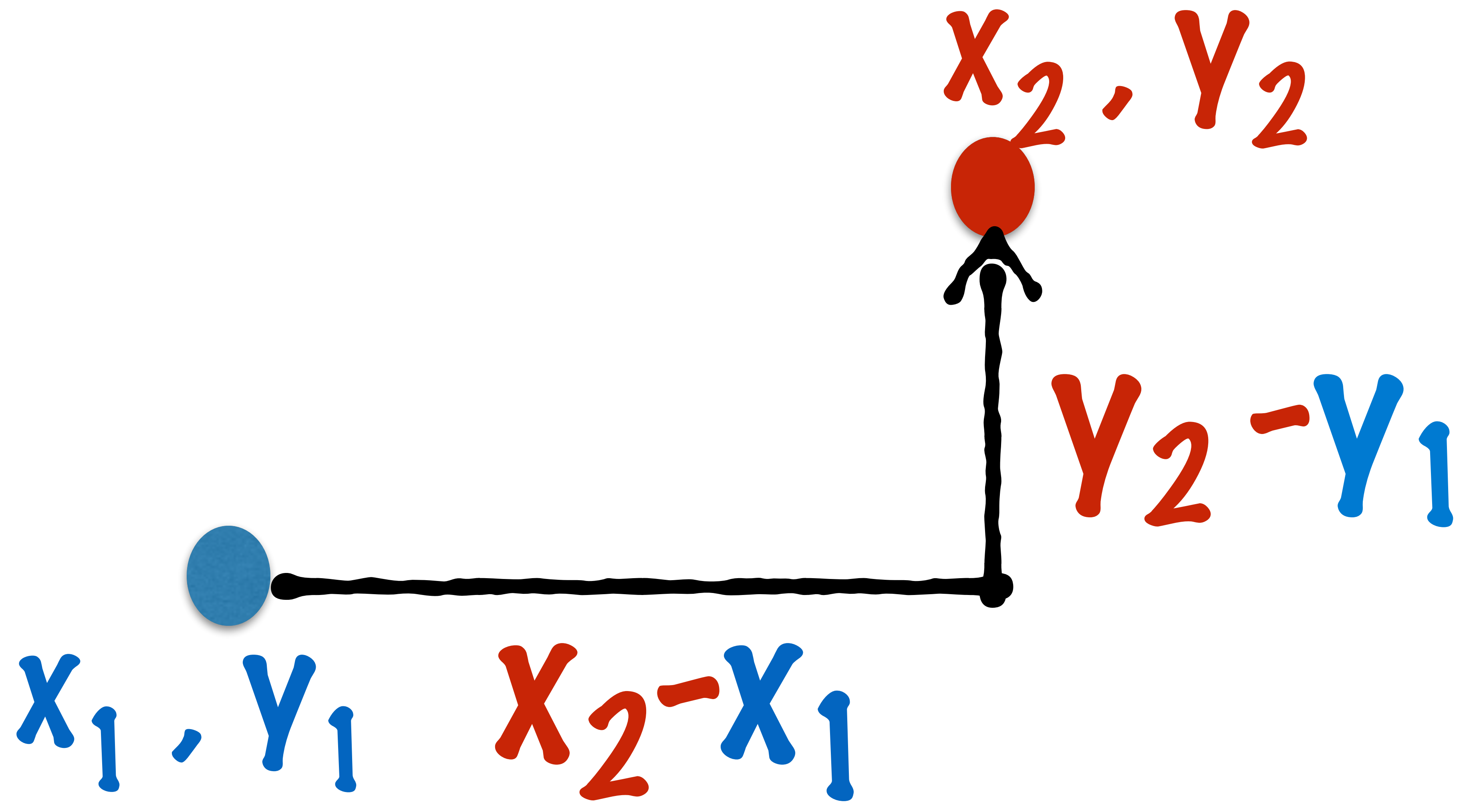
One simple example is the Euclidean distance



K-MEANS CLUSTERING

Distance Metrics

Another example is the **Manhattan distance**



$$d\,(x\,y) = |x_1 - x_2| + |y_1 - y_2|$$

K-MEANS CLUSTERING

There are many possible **Distance Metrics** we could choose from

Euclidean
Manhattan
Haversine
Cosine
Jaccard

`double[ ]`

p1	3	2.5
p2	1	4
p3	2	3
p4	4	5.5
p5	6	6

K-MEANS CLUSTERING

It's clear from both our examples of Distance metrics

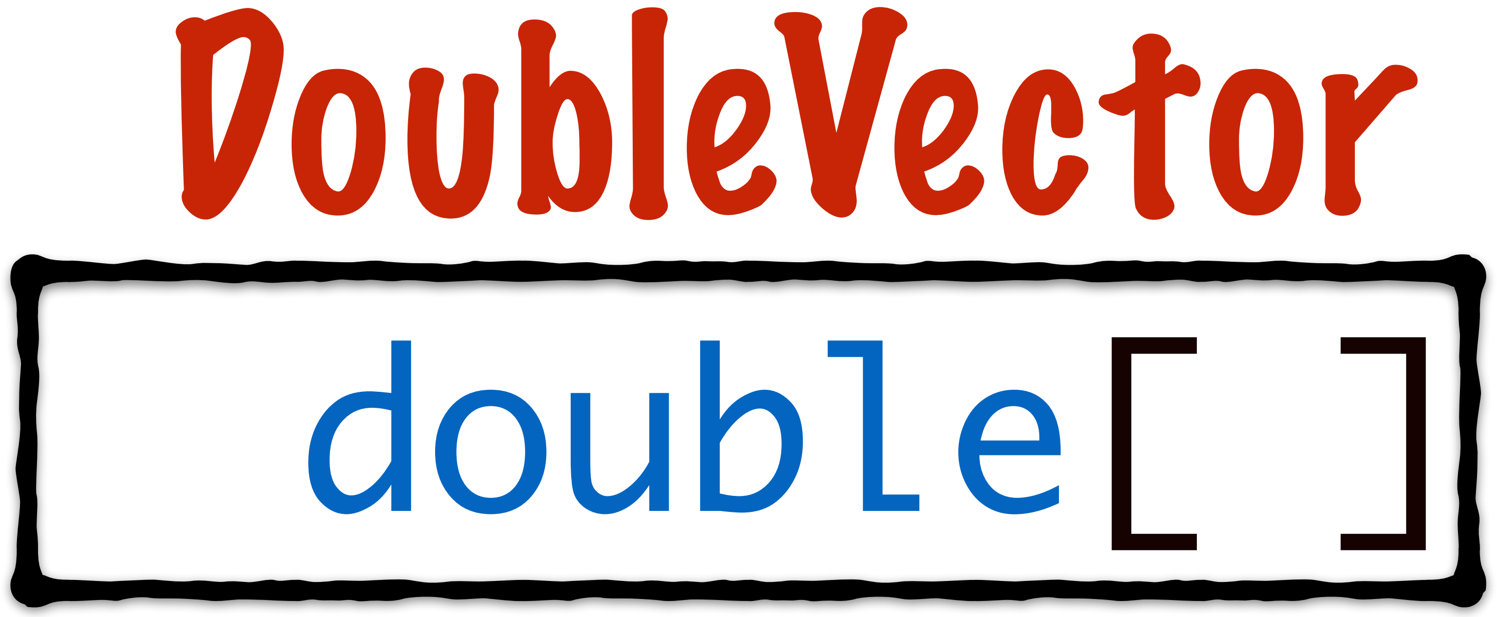
We'll need to perform a bunch of mathematical operations on these arrays

`double[ ]`

p1	3	2.5
p2	1	4
p3	2	3
p4	4	5.5
p5	6	6

K-MEANS CLUSTERING

We define an interface **DoubleVector** which has a bunch of methods for mathematical operations

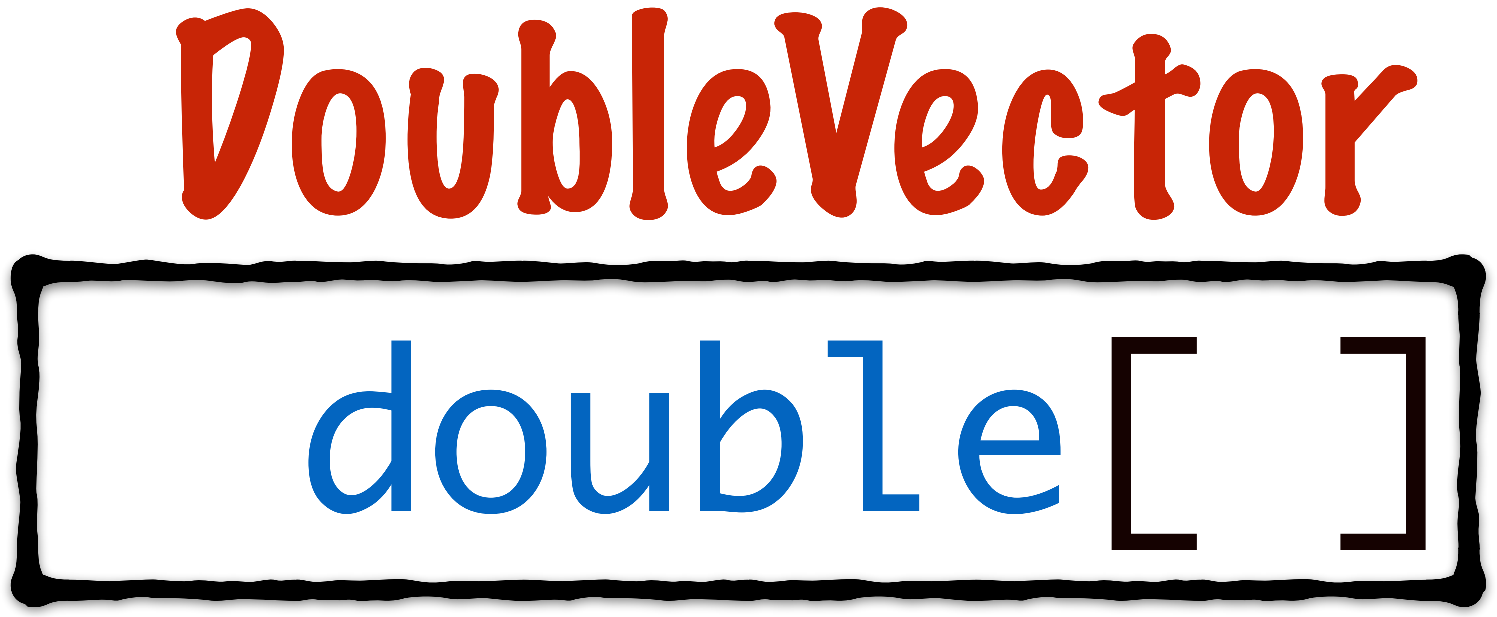


DoubleVector

double[ ]

K-MEANS CLUSTERING

We define an interface DoubleVector which has a bunch of methods for mathematical operations

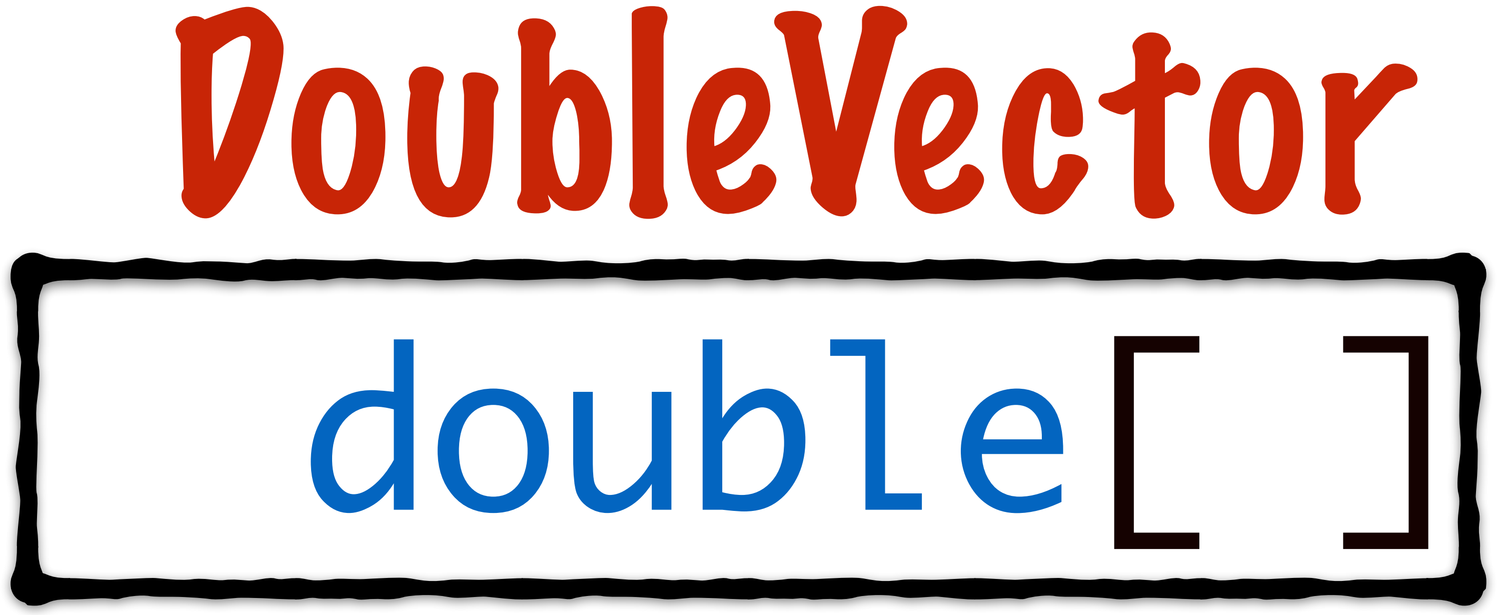


DoubleVector

double[ ]

```
public interface DoubleVector {
}
```

K-MEANS CLUSTERING

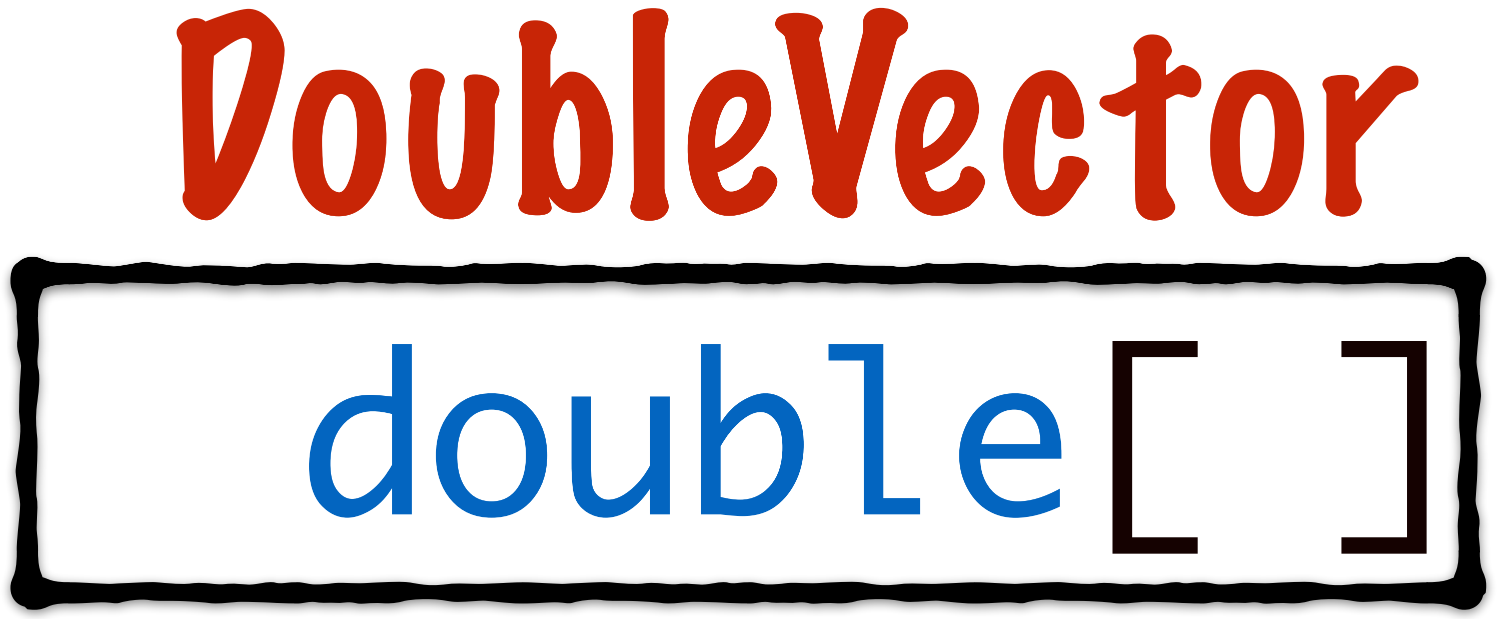


```
public interface DoubleVector {
  add()            multiply ()
  subtract ()        pow ()
  divide ()          abs ()
}
```

..and many others

K-MEANS CLUSTERING

DenseDoubleVector is an implementation of DoubleVector



DoubleVector

double[ ]

```
public final class DenseDoubleVector implements
DoubleVector {

private final double[] vector;
```

It has one member variable

K-MEANS CLUSTERING

DenseDoubleVector is an implementation of DoubleVector

DoubleVector

double[ ]

```
public final class DenseDoubleVector implements DoubleVector {
private final double[] vector;
```

It also has implementations for many mathematical methods, here are a couple

```
public final class DenseDoubleVector implements
DoubleVector {
private final double[] vector;

@Override
public double sum() {
   double sum = 0.0d;
   for (double aVector : vector)
{
      sum += aVector;
   }
   return sum;
}
```

This method returns the sum of all the elements of the vector

In math, arrays are more popularly known as vectors

```
public final class DenseDoubleVector implements
DoubleVector {
private final double[] vector;

@Override
public DoubleVector abs() {
  DoubleVector v = new
DenseDoubleVector(getLength());
  for (int i = 0; i < v.getLength(); i
++) {
    v.set(i, FastMath.abs(vector[i]));
  }
  return v;
}
```

This method returns a vector with the absolute values of our vector's elements

```
public final class DenseDoubleVector implements
DoubleVector {
private final double[] vector;

@Override
public DoubleVector abs() {
  DoubleVector v = new
DenseDoubleVector(getLength());
  for (int i = 0; i < v.getLength(); i
++) {
    v.set(i, FastMath.abs(vector[i]));
  }
  return v;
}
```

This is from **Apache Commons Math**

```java
public final class DenseDoubleVector implements
DoubleVector {
private final double[] vector;

@Override
public DoubleVector abs() {
  DoubleVector v = new
DenseDoubleVector(getLength());
  for (int i = 0; i < v.getLength(); i
++) {
    v.set(i, FastMath.abs(vector[i]));
  }
  return v;
}
```

Apache Commons Math

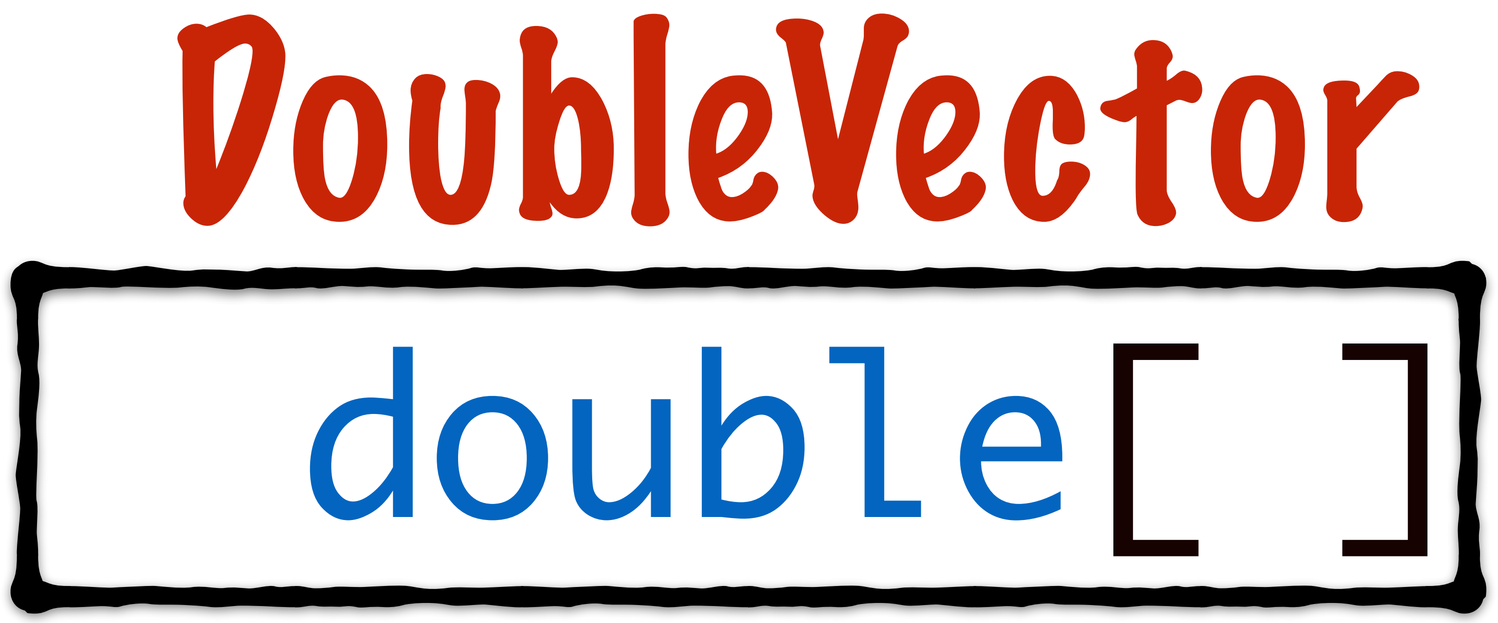
which is a very lightweight Math library

```java
public final class DenseDoubleVector implements
DoubleVector {
private final double[] vector;
```

There are many other methods also implemented

You can check out the source code for more details

DistanceMeasurer is an interface for defining a distance metric



DoubleVector

double[ ]

```
public interface DistanceMeasurer {

  public double measureDistance(double[] set1, double[] set2);

  public double measureDistance(DoubleVector vec1,
DoubleVector vec2);

}
```

DistanceMeasurer

```
public interface DistanceMeasurer {

  public double measureDistance(double[] set1, double[] set2);

  public double measureDistance(DoubleVector vec1,
DoubleVector vec2);

}
```

measureDistance() is the method for measuring distance

DistanceMeasurer

ManhattanDistance
implements
DistanceMeasurer

DistanceMeasurer

ManhattanDistance

```java
public final class ManhattanDistance implements DistanceMeasurer {

  @Override
  public double measureDistance(double[] set1, double[] set2) {
    double sum = 0;
    int length = set1.length;
    for (int i = 0; i < length; i++) {
      sum += Math.abs(set1[i] - set2[i]);
    }
    return sum;
  }

  @Override
  public double measureDistance(DoubleVector vec1, DoubleVector vec2) {
    return vec1.subtract(vec2).abs().sum();
  }
```

DistanceMeasurer

ManhattanDistance

```java
public final class ManhattanDistance implements DistanceMeasurer {

   @Override
   public double measureDistance(double[] set1, double[] set2) {
      double sum = 0;
      int length = set1.length;
      for (int i = 0; i < length; i++) {
       sum += Math.abs(set1[i] - set2[i]);
      }
      return sum;
   }

   @Override
   public double measureDistance(DoubleVector vec1, DoubleVector vec2) {
      return vec1.subtract(vec2).abs().sum();
   }
}
```

measureDistance() implemented for double[] arrays

DistanceMeasurer

ManhattanDistance

```java
public final class ManhattanDistance implements DistanceMeasurer {

   @Override
   public double measureDistance(double[] set1, double[] set2) {
      double sum = 0;
      int length = set1.length;
      for (int i = 0; i < length; i++) {
         sum += Math.abs(set1[i] - set2[i]);
      }
      return sum;
   }

   @Override
   public double measureDistance(DoubleVector vec1, DoubleVector vec2) {
      return vec1.subtract(vec2).abs().sum();
   }

}
```

$$d\,(x\,y) = |x_1 - x_2| + |y_1 - y_2|$$

DistanceMeasurer

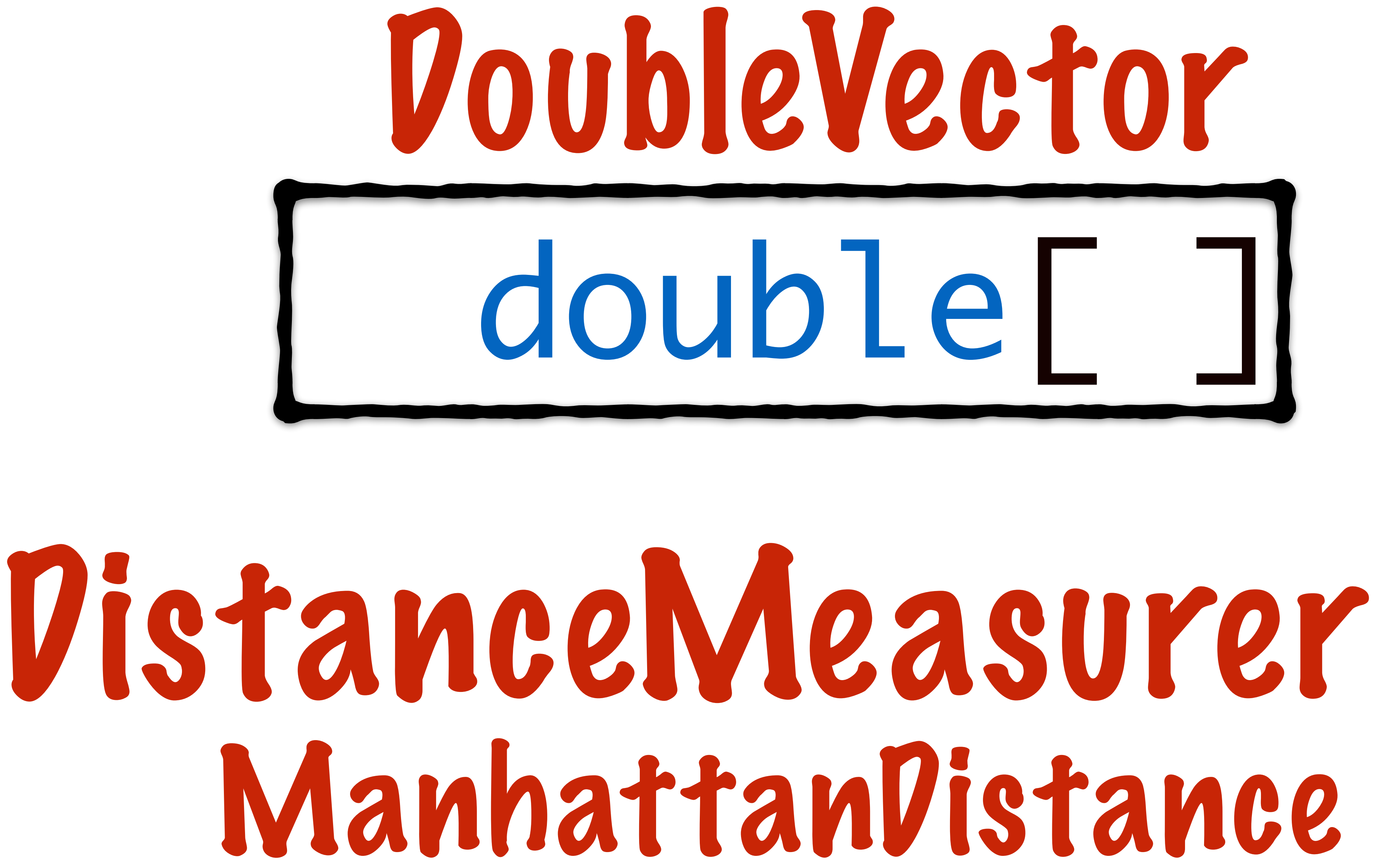
ManhattanDistance

```
public final class ManhattanDistance implements DistanceMeasurer {

   @Override
   public double measureDistance(double[] set1, double[] set2) {
      double sum = 0;
      int length = set1.length;
      for (int i = 0; i < length; i++) {
         sum += Math.abs(set1[i] - set2[i]);
      }
      return sum;
   }
```

$$d\,(x\;y) = |x_1 - x_2| + |y_1 - y_2|$$

```
  @Override
  public double measureDistance(DoubleVector vec1, DoubleVector
vec2) {
    return vec1.subtract(vec2).abs().sum();
  }

}
```

There are several other DistanceMeasurer implementations available in the source code


DoubleVector

double[ ]

DistanceMeasurer

ManhattanDistance

Now let's get to the Hadoop classes we'll need

DoubleVector

double[ ]

DistanceMeasurer

ManhattanDistance

We'll need a Writable wrapper for the DoubleVector

DoubleVector

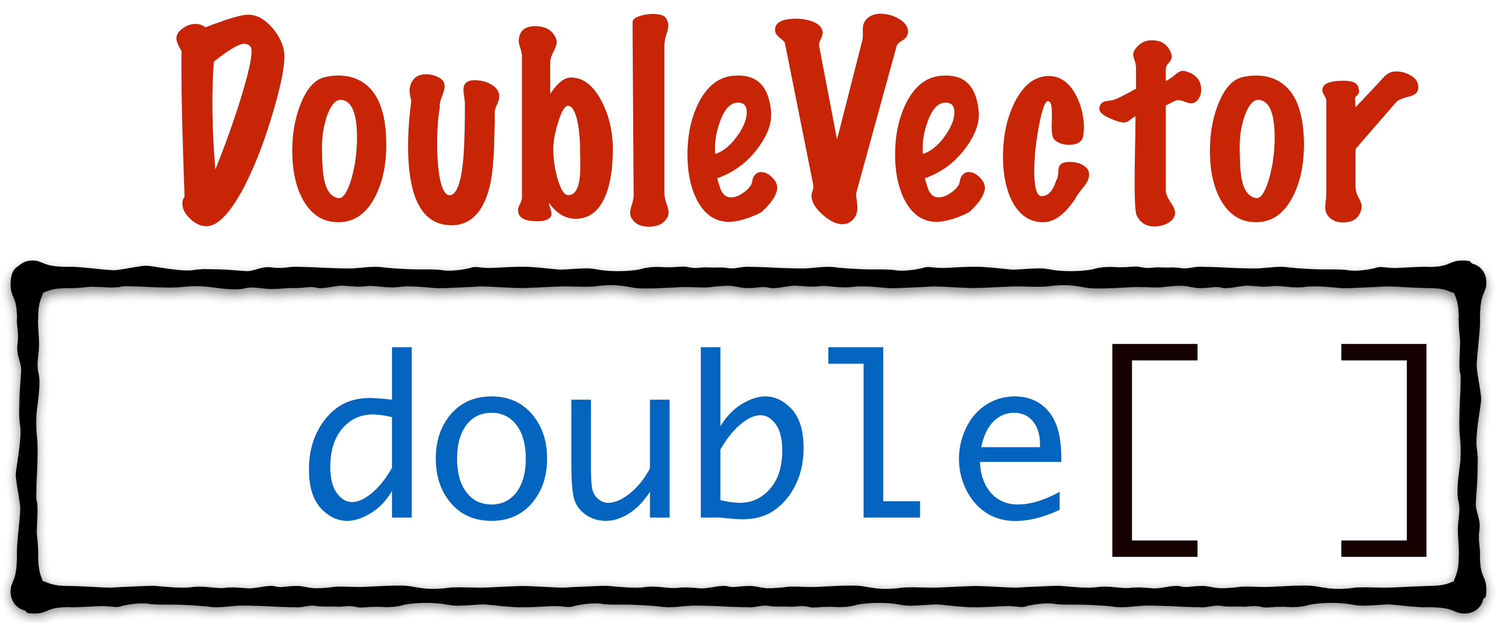
`double[ ]`

DistanceMeasurer

ManhattanDistance

We'll actually write 2 wrappers though

VectorWritable which represents any data point



DoubleVector

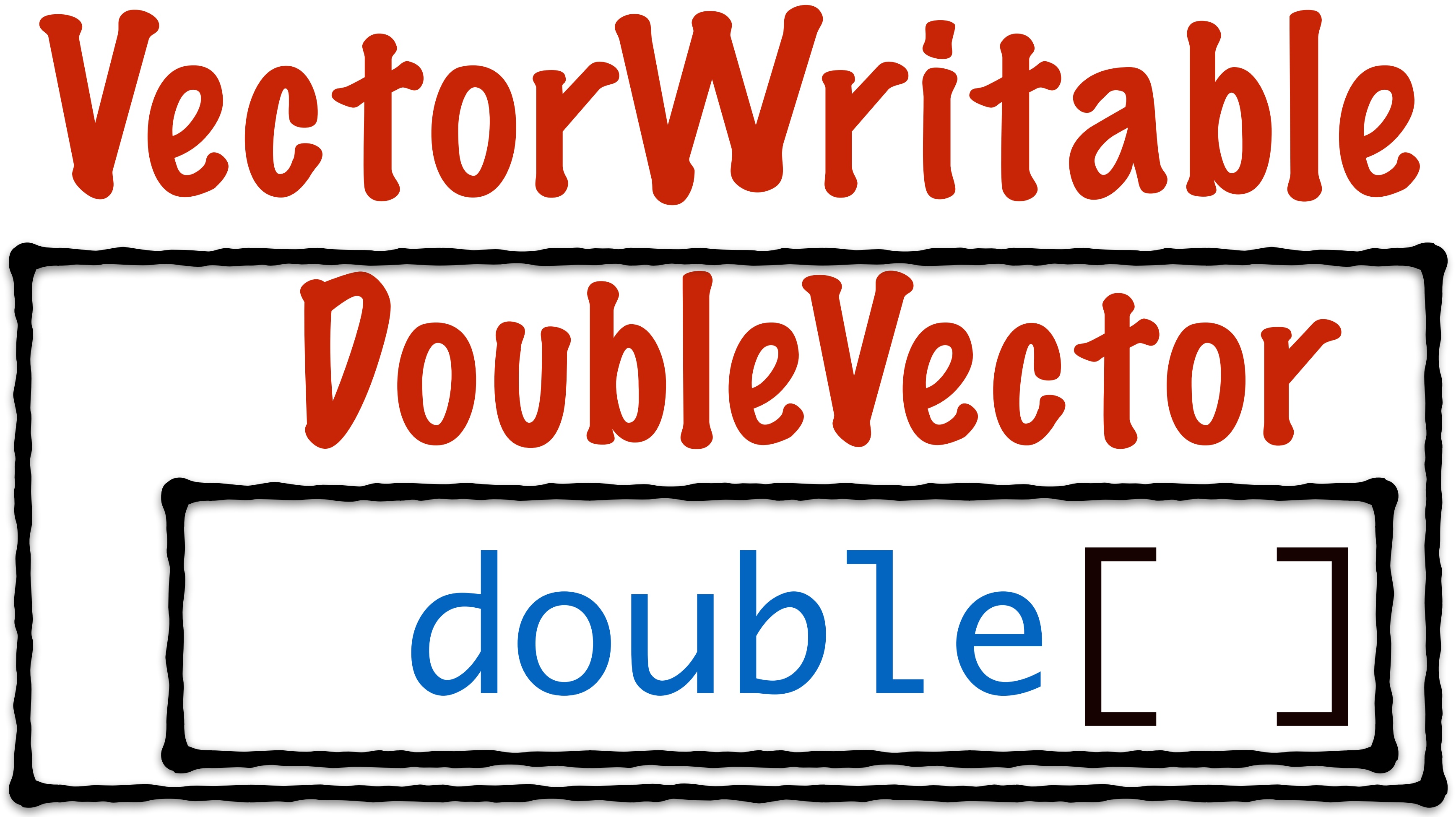
double[ ]

DistanceMeasurer

ManhattanDistance

We'll actually write 2 wrappers though

ClusterCenter which represents the center of cluster



VectorWritable

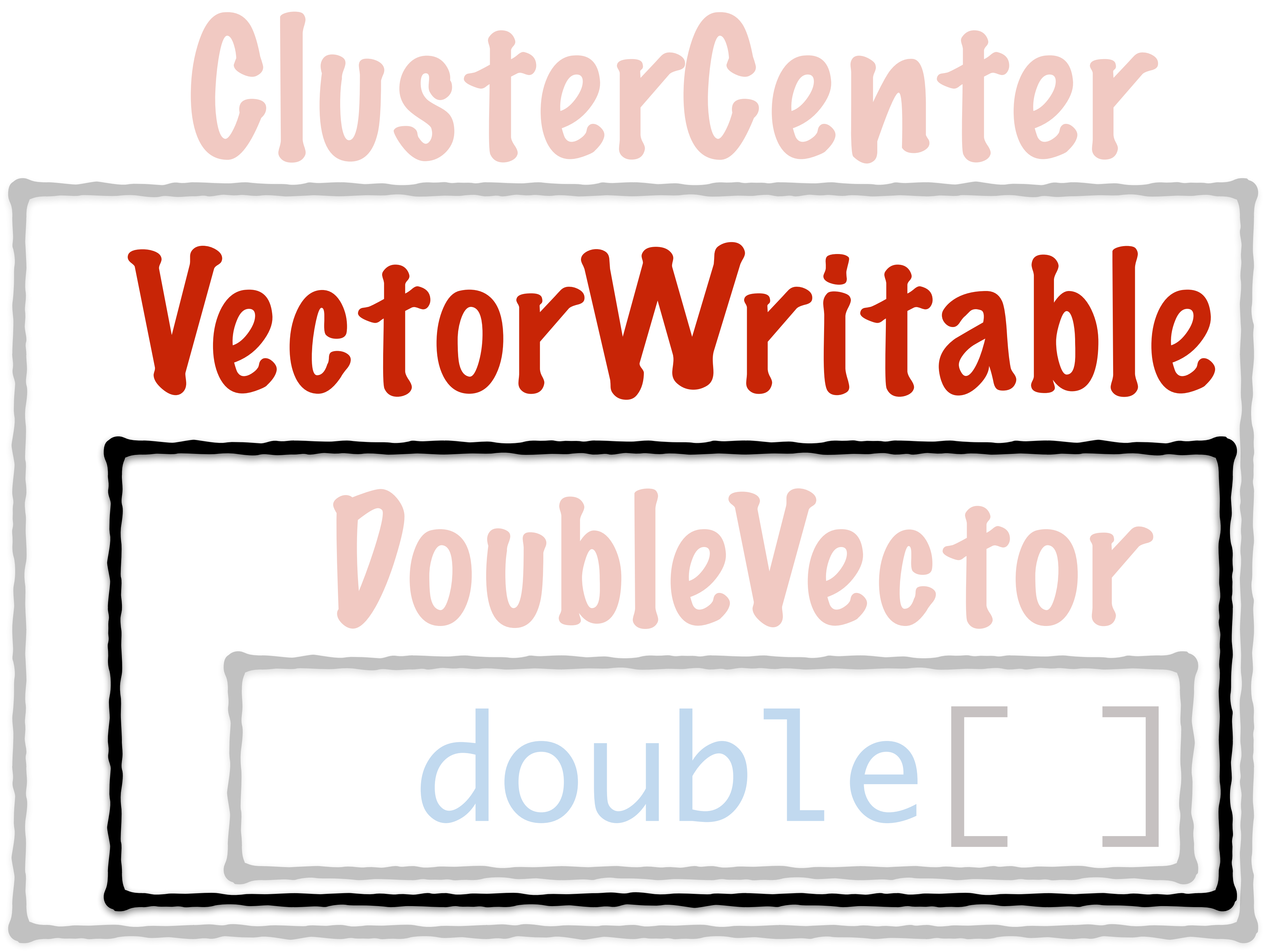
DoubleVector

double[ ]

DistanceMeasurer

ManhattanDistance

```
public final class
VectorWritable
implements
WritableComparable
<VectorWritable> {
```



ClusterCenter

VectorWritable

DoubleVector

double[ ]

DistanceMeasurer

ManhattanDistance

VectorWritable

```
public final class VectorWritable implements
WritableComparable<VectorWritable> {

private DoubleVector vector;
```

This class has 1 member

Let's look at the implementations of readFields(), write() and compareTo()

VectorWritable

```
public final class VectorWritable implements
WritableComparable<VectorWritable> {
private DoubleVector vector;

@Override
public final void write(DataOutput out) throws IOException {
  writeVector(this.vector, out);
}

@Override
public final void readFields(DataInput in) throws IOException {
  this.vector = readVector(in);
}

@Override
public final int compareTo(VectorWritable o) {
  return compareVector(this, o);
}
```

VectorWritable

```java
public final class VectorWritable implements
WritableComparable<VectorWritable> {
private DoubleVector vector;

@Override
public final void write(DataOutput out) throws IOException {
  writeVector(this.vector, out);
}

@Override
public final void readFields(DataInput in) throws IOException {
  this.vector = readVector(in);
}

@Override
public final int compareTo(VectorWritable o) {
  return compareVector(this, o);
}
```

Each of these calls another method which will also be implemented for DoubleVector

VectorWritable

```
public final class VectorWritable implements
WritableComparable<VectorWritable> {
private DoubleVector vector;

@Override
public final void write(DataOutput out) throws IOException {
  writeVector(this.vector, out);
}

@Override
public final void readFields(DataInput in) throws IOException {
  this.vector = readVector(in);
}

@Override
public final int compareTo(VectorWritable o) {
  return compareVector(this, o);
}
```

VectorWritable

```
public final class VectorWritable implements
WritableComparable<VectorWritable> {
private DoubleVector vector;
```

readVector()

```
    public static DoubleVector readVector(DataInput in)
    throws IOException {
      final int length = in.readInt();
      DoubleVector vector = new DenseDoubleVector(length);
      for (int i = 0; i < length; i++) {
        vector.set(i, in.readDouble());
      }
      return vector;
    }
```

Instantiates a DenseDoubleVector and adds the input data to it

VectorWritable

```
public final class VectorWritable implements
WritableComparable<VectorWritable> {
private DoubleVector vector;
```

writeVector()

```
  public static void writeVector(DoubleVector
  vector, DataOutput out) throws IOException {
    out.writeInt(vector.getLength());
    for (int i = 0; i < vector.getDimension(); i++) {
      out.writeDouble(vector.get(i));
    }
  }
```

Writes out each element of the vector

VectorWritable

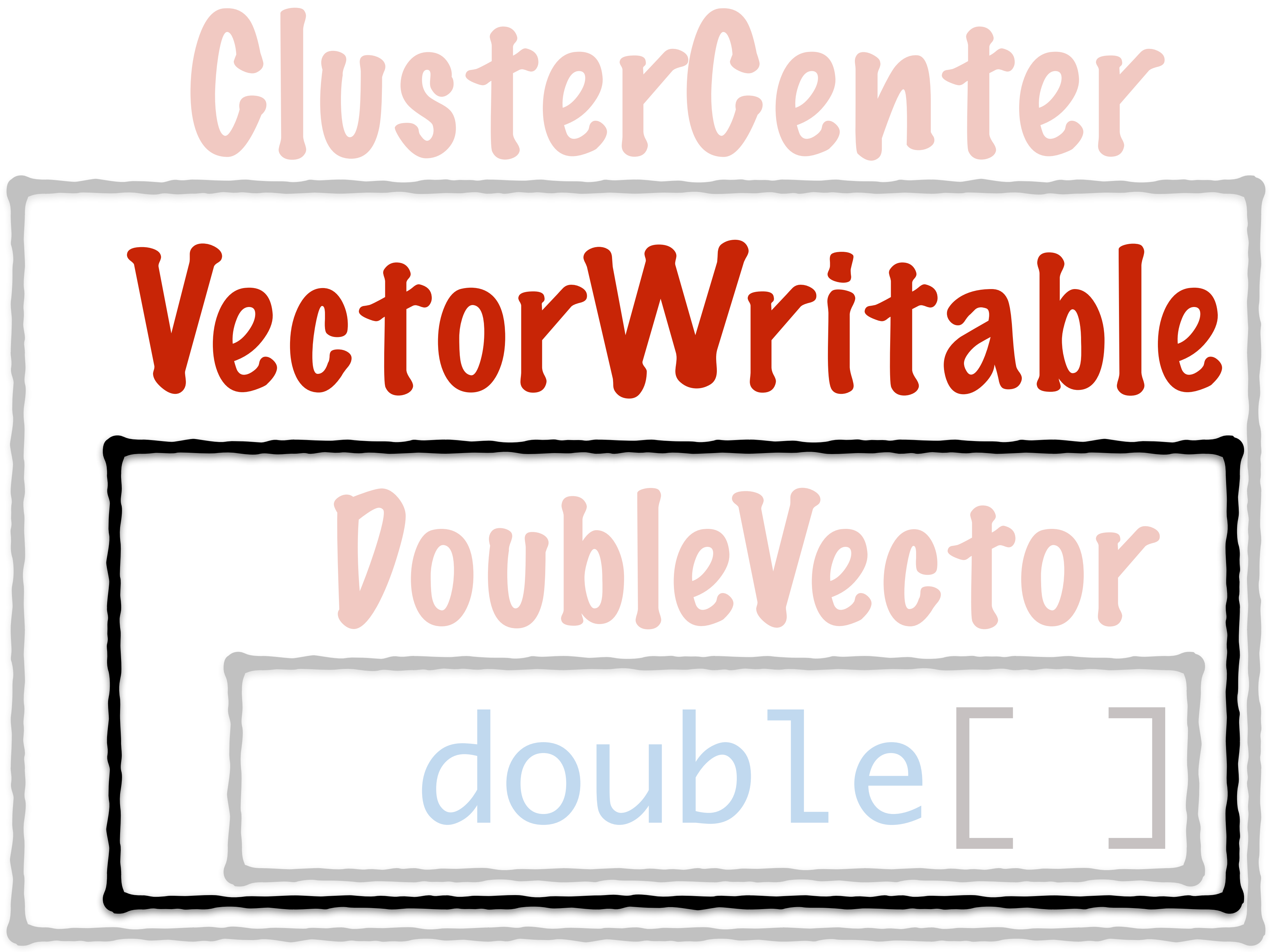
```
public final class VectorWritable implements
WritableComparable<VectorWritable> {
private DoubleVector vector;
```

compareVector()

```
public static int compareVector(DoubleVector a, DoubleVector
o) {
  DoubleVector subtract = a.subtract(o);
  return (int) subtract.sum();
}
```

Returns the sign of the sum of differences of corresponding elements

```
public final class
VectorWritable
implements
WritableComparable
<VectorWritable> {
```



ClusterCenter

VectorWritable

DoubleVector

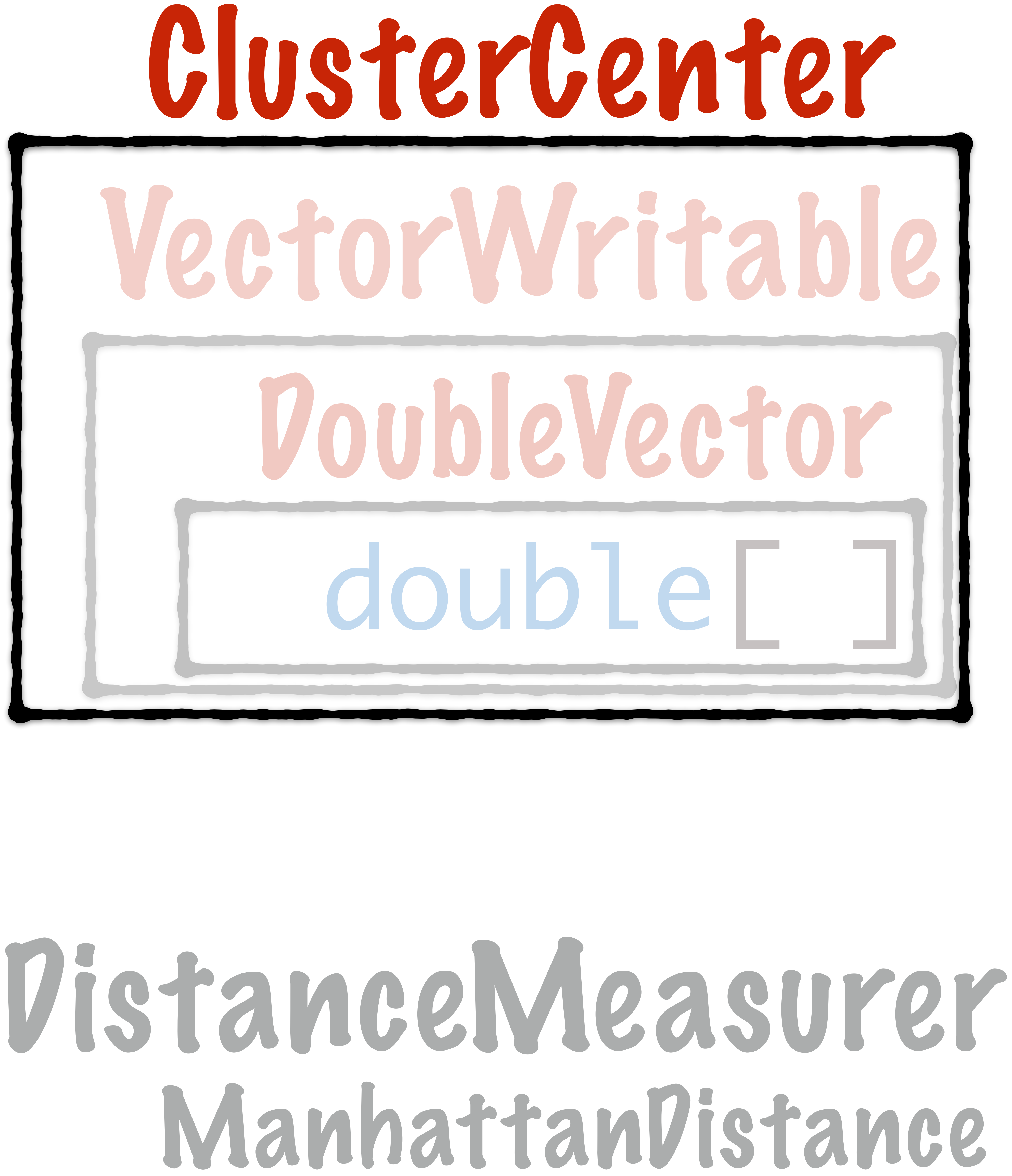
double[ ]

DistanceMeasurer

ManhattanDistance

ClusterCenter is very similar to VectorWritable

It just has a couple of extra methods to measure how close we are to convergence



ClusterCenter

```
public final class ClusterCenter
 implements WritableComparable<ClusterCenter> {

 private DoubleVector center;


public final double calculateError(DoubleVector v) {
  return Math.sqrt(center.subtract(v).abs().sum());
}
```

Let's just look at the extra methods in ClusterCenter

This method compares another DoubleVector with the ClusterCenter

ClusterCenter

```java
public final class ClusterCenter
 implements WritableComparable<ClusterCenter> {

 private DoubleVector center;




public final double calculateError(DoubleVector v) {
  return Math.sqrt(center.subtract(v).abs().sum());
}
```

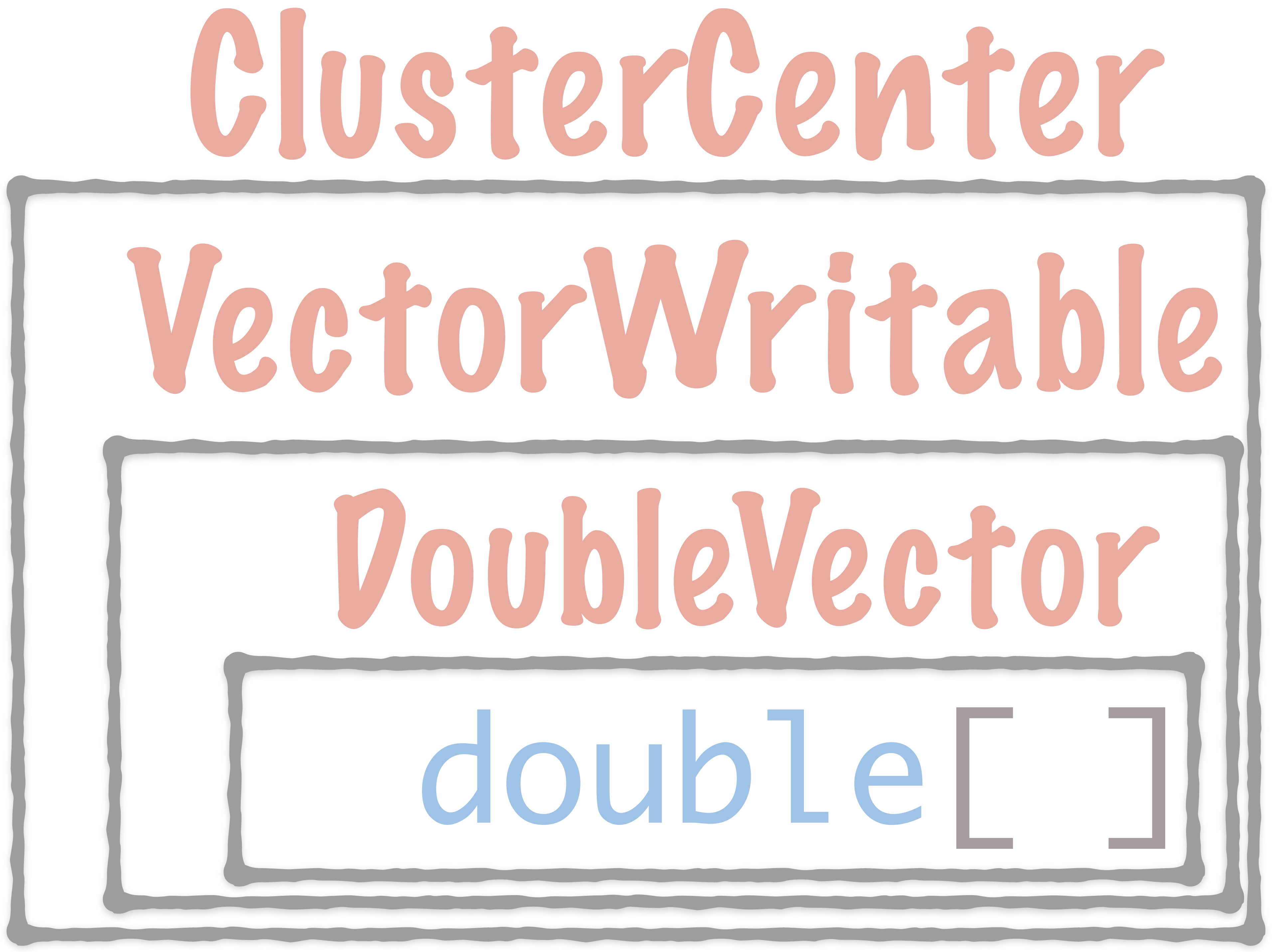
This is used to calculate a difference between the old and new cluster centers

ClusterCenter

```
public final class ClusterCenter
 implements WritableComparable<ClusterCenter> {

 private DoubleVector center;


 public final boolean converged(ClusterCenter c) {
  return calculateError(c.getCenterVector()) > 0;
 }
```

Converged returns a boolean that's true if the 2 cluster centers are different

Let's now look at the Main Job


ClusterCenter

VectorWritable

DoubleVector

double[ ]

DistanceMeasurer

ManhattanDistance

KMeansClusteringJob

```java
public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {

    int iteration = 1;
    Configuration conf = new Configuration();
    conf.set("num.iteration", iteration + "");

    Path in = new Path("files/clustering/import/data");
    Path center = new Path("files/clustering /import/center/cen.seq");
    conf.set("centroid.path", center.toString());
    Path out = new Path("files/clustering/depth_1");

    Job job = Job.getInstance(conf);
    job.setJobName("KMeans Clustering");

    job.setMapperClass(KMeansMapper.class);
    job.setReducerClass(KMeansReducer.class);
    job.setJarByClass(KMeansMapper.class);

    FileInputFormat.addInputPath(job, in);
    FileSystem fs = FileSystem.get(conf);
    if (fs.exists(out)) {
        fs.delete(out, true);
    }

    if (fs.exists(center)) {
        fs.delete(out, true);
    }

    if (fs.exists(in)) {
        fs.delete(in, true);
    }

    writeExampleCenters(conf, center, fs);

    writeExampleVectors(conf, in, fs);

    FileOutputFormat.setOutputPath(job, out);
    job.setInputFormatClass(SequenceFileInputFormat.class);
    job.setOutputFormatClass(SequenceFileOutputFormat.class);

    job.setOutputKeyClass(ClusterCenter.class);
    job.setOutputValueClass(VectorWritable.class);

    job.waitForCompletion(true);

    long counter = job.getCounters().findCounter(KMeansReducer.Counter.CONVERGED).getValue();
    iteration++;
```

We set an iteration counter to 1

KMeansClusteringJob

```java
public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {

    int iteration = 1;
    Configuration conf = new Configuration();
    conf.set("num.iteration", iteration + "");

    Path in = new Path("files/clustering/import/data");
    Path center = new Path("files/clustering /import/center/cen.seq");
    conf.set("centroid.path", center.toString());
    Path out = new Path("files/clustering/depth_1");

    Job job = Job.getInstance(conf);
    job.setJobName("KMeans Clustering");

    job.setMapperClass(KMeansMapper.class);
    job.setReducerClass(KMeansReducer.class);
    job.setJarByClass(KMeansMapper.class);

    FileInputFormat.addInputPath(job, in);
    FileSystem fs = FileSystem.get(conf);
    if (fs.exists(out)) {
        fs.delete(out, true);
    }

    if (fs.exists(center)) {
        fs.delete(out, true);
    }

    if (fs.exists(in)) {
        fs.delete(in, true);
    }

    writeExampleCenters(conf, center, fs);

    writeExampleVectors(conf, in, fs);

    FileOutputFormat.setOutputPath(job, out);
    job.setInputFormatClass(SequenceFileInputFormat.class);
    job.setOutputFormatClass(SequenceFileOutputFormat.class);

    job.setOutputKeyClass(ClusterCenter.class);
    job.setOutputValueClass(VectorWritable.class);

    job.waitForCompletion(true);

    long counter = job.getCounters().findCounter(KMeansReducer.Counter.CONVERGED).getValue();
```

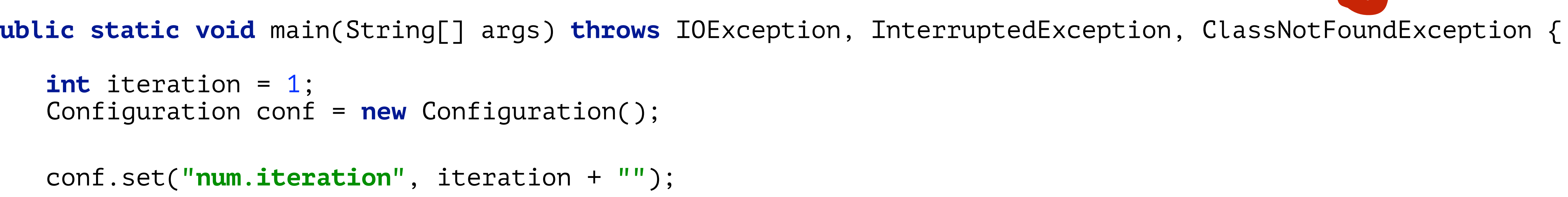
We set an iteration counter to 1

KMeansClusteringJob

```java
public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {

    int iteration = 1;
    Configuration conf = new Configuration();
    conf.set("num.iteration", iteration + "");

    Path in = new Path("files/clustering/import/data");
    Path center = new Path("files/clustering /import/center/cen.seq");
    conf.set("centroid.path", center.toString());

    Path out = new Path("files/clustering/depth_1");


    Job job = Job.getInstance(conf);
    job.setJobName("KMeans Clustering");

    job.setMapperClass(KMeansMapper.class);
    job.setReducerClass(KMeansReducer.class);
    job.setJarByClass(KMeansMapper.class);

    FileInputFormat.addInputPath(job, in);
    FileSystem fs = FileSystem.get(conf);
    if (fs.exists(out)) {
        fs.delete(out, true);
    }

    if (fs.exists(center)) {
        fs.delete(out, true);
    }

    if (fs.exists(in)) {
        fs.delete(in, true);
    }

    writeExampleCenters(conf, center, fs);

    writeExampleVectors(conf, in, fs);

    FileOutputFormat.setOutputPath(job, out);
    job.setInputFormatClass(SequenceFileInputFormat.class);
    job.setOutputFormatClass(SequenceFileOutputFormat.class);

    job.setOutputKeyClass(ClusterCenter.class);
    job.setOutputValueClass(VectorWritable.class);
```

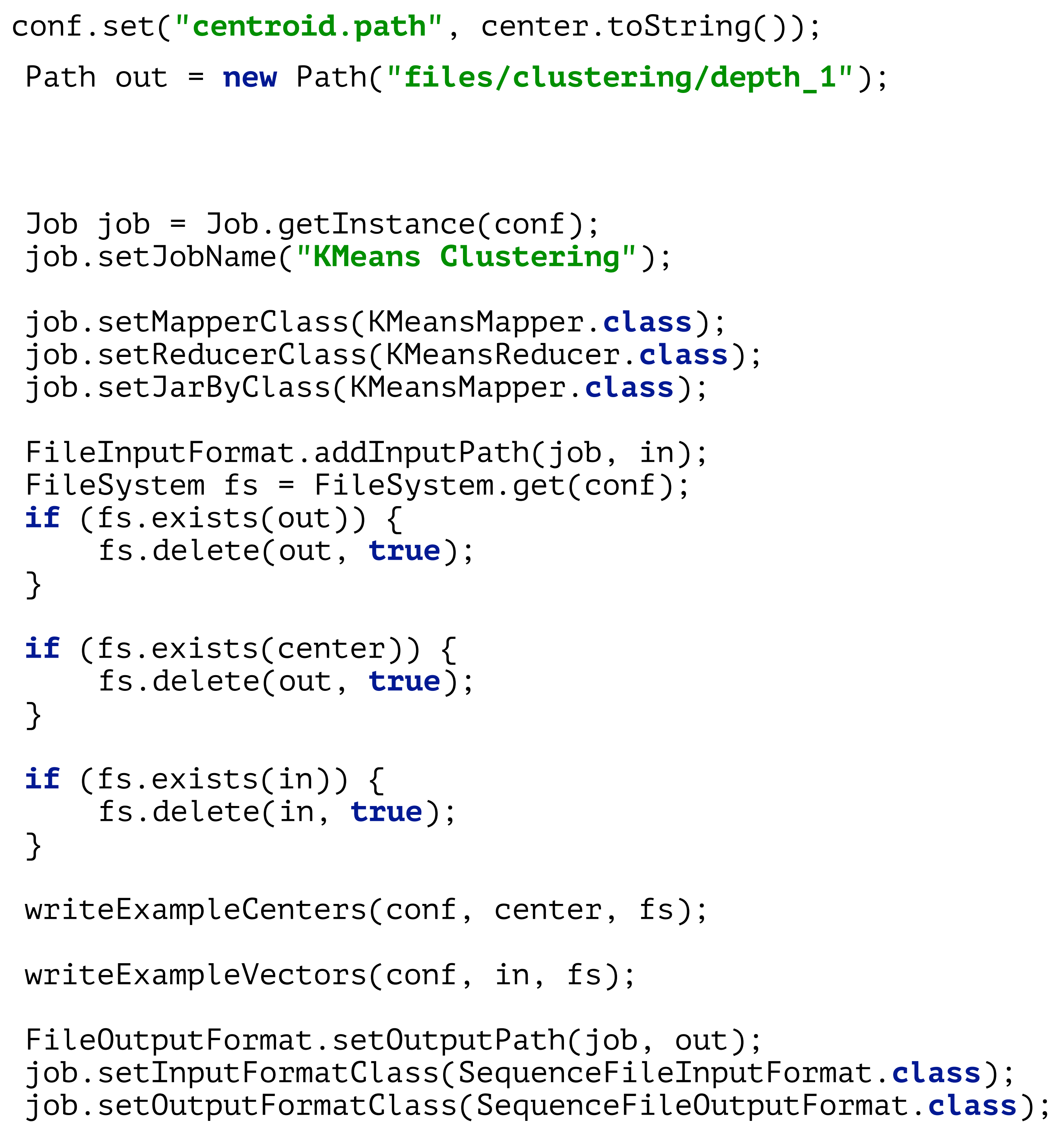
We set iteration number as a parameter in the Job configuration

KMeansClusteringJob



```
ublic static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {

   int iteration = 1;
   Configuration conf = new Configuration();

   conf.set("num.iteration", iteration + "");
```

```
Path in = new Path("files/clustering/import/data");
Path center = new Path("files/clustering /import/center/cen.seq"
```



```
  conf.set("centroid.path", center.toString());
   Path out = new Path("files/clustering/depth_1");


   Job job = Job.getInstance(conf);
   job.setJobName("KMeans Clustering");

   job.setMapperClass(KMeansMapper.class);
   job.setReducerClass(KMeansReducer.class);
   job.setJarByClass(KMeansMapper.class);

   FileInputFormat.addInputPath(job, in);
   FileSystem fs = FileSystem.get(conf);
   if (fs.exists(out)) {
       fs.delete(out, true);
   }

   if (fs.exists(center)) {
       fs.delete(out, true);
   }

   if (fs.exists(in)) {
       fs.delete(in, true);
   }

   writeExampleCenters(conf, center, fs);

   writeExampleVectors(conf, in, fs);

   FileOutputFormat.setOutputPath(job, out);
   job.setInputFormatClass(SequenceFileInputFormat.class);
   job.setOutputFormatClass(SequenceFileOutputFormat.class);
```

The file paths for the input data and the initial centers

KMeansClusteringJob

```
ublic static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {

   int iteration = 1;
   Configuration conf = new Configuration();

   conf.set("num.iteration", iteration + "");


Path in = new Path("files/clustering/import/data");
Path center = new Path("files/clustering /import/center/cen.seq"


  conf.set("centroid.path", center.toString());
   Path out = new Path("files/clustering/depth_1");


   Job job = Job.getInstance(conf);
   job.setJobName("KMeans Clustering");

   job.setMapperClass(KMeansMapper.class);
   job.setReducerClass(KMeansReducer.class);
   job.setJarByClass(KMeansMapper.class);

   FileInputFormat.addInputPath(job, in);
   FileSystem fs = FileSystem.get(conf);
   if (fs.exists(out)) {
       fs.delete(out, true);
   }

   if (fs.exists(center)) {
       fs.delete(out, true);
   }

   if (fs.exists(in)) {
       fs.delete(in, true);
   }

   writeExampleCenters(conf, center, fs);

   writeExampleVectors(conf, in, fs);

   FileOutputFormat.setOutputPath(job, out);
   job.setInputFormatClass(SequenceFileInputFormat.class);
   job.setOutputFormatClass(SequenceFileOutputFormat.class);
```

We have a couple of methods in this class that can write some sample data to these paths

KMeansClusteringJob

```
ublic static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {

   int iteration = 1;
   Configuration conf = new Configuration();

   conf.set("num.iteration", iteration + "");


   Path in = new Path("files/clustering/import/data");

   Path center = new Path("files/clustering /import/center/cen.seq");



  conf.set("centroid.path", center.toString());
   Path out = new Path("files/clustering/depth_1");


   Job job = Job.getInstance(conf);
   job.setJobName("KMeans Clustering");

   job.setMapperClass(KMeansMapper.class);
   job.setReducerClass(KMeansReducer.class);
   job.setJarByClass(KMeansMapper.class);

   FileInputFormat.addInputPath(job, in);
   FileSystem fs = FileSystem.get(conf);
   if (fs.exists(out)) {
       fs.delete(out, true);
   }

   if (fs.exists(center)) {
       fs.delete(out, true);
   }

   if (fs.exists(in)) {
       fs.delete(in, true);
   }

   writeExampleCenters(conf, center, fs);

   writeExampleVectors(conf, in, fs);

   FileOutputFormat.setOutputPath(job, out);
   job.setInputFormatClass(SequenceFileInputFormat.class);
   job.setOutputFormatClass(SequenceFileOutputFormat.class);
```

In every iteration, we go through all the input data points again

KMeansClusteringJob

```java
public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {

    int iteration = 1;
    Configuration conf = new Configuration();

    conf.set("num.iteration", iteration + "");


    Path in = new Path("files/clustering/import/data");
    Path center = new Path("files/clustering /import/center/cen.seq");
    conf.set("centroid.path", center.toString());

    Path out = new Path("files/clustering/depth_1");


    Job job = Job.getInstance(conf);
    job.setJobName("KMeans Clustering");

    job.setMapperClass(KMeansMapper.class);
    job.setReducerClass(KMeansReducer.class);
    job.setJarByClass(KMeansMapper.class);

    FileInputFormat.addInputPath(job, in);
    FileSystem fs = FileSystem.get(conf);
    if (fs.exists(out)) {
        fs.delete(out, true);
    }

    if (fs.exists(center)) {
        fs.delete(out, true);
    }

    if (fs.exists(in)) {
        fs.delete(in, true);
    }

    writeExampleCenters(conf, center, fs);

    writeExampleVectors(conf, in, fs);

    FileOutputFormat.setOutputPath(job, out);
    job.setInputFormatClass(SequenceFileInputFormat.class);
    job.setOutputFormatClass(SequenceFileOutputFormat.class);

    job.setOutputKeyClass(ClusterCenter.class);
    job.setOutputValueClass(VectorWritable.class);
```

The centers are also an input to the mapper

KMeansClusteringJob

```
public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {

    int iteration = 1;
    Configuration conf = new Configuration();

    conf.set("num.iteration", iteration + "");


    Path in = new Path("files/clustering/import/data");
    Path center = new Path("files/clustering /import/center/cen.seq");
    conf.set("centroid.path", center.toString());

    Path out = new Path("files/clustering/depth_1");


    Job job = Job.getInstance(conf);
    job.setJobName("KMeans Clustering");

    job.setMapperClass(KMeansMapper.class);
    job.setReducerClass(KMeansReducer.class);
    job.setJarByClass(KMeansMapper.class);

    FileInputFormat.addInputPath(job, in);
    FileSystem fs = FileSystem.get(conf);
    if (fs.exists(out)) {
        fs.delete(out, true);
    }

    if (fs.exists(center)) {
        fs.delete(out, true);
    }

    if (fs.exists(in)) {
        fs.delete(in, true);
    }

    writeExampleCenters(conf, center, fs);

    writeExampleVectors(conf, in, fs);

    FileOutputFormat.setOutputPath(job, out);
    job.setInputFormatClass(SequenceFileInputFormat.class);
    job.setOutputFormatClass(SequenceFileOutputFormat.class);

    job.setOutputKeyClass(ClusterCenter.class);
    job.setOutputValueClass(VectorWritable.class);
```

These will change in each iteration though

KMeansClusteringJob

```
public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {

    int iteration = 1;
    Configuration conf = new Configuration();

    conf.set("num.iteration", iteration + "");


    Path in = new Path("files/clustering/import/data");
    Path center = new Path("files/clustering /import/center/cen.seq");
    conf.set("centroid.path", center.toString());

    Path out = new Path("files/clustering/depth_1");


    Job job = Job.getInstance(conf);
    job.setJobName("KMeans Clustering");

    job.setMapperClass(KMeansMapper.class);
    job.setReducerClass(KMeansReducer.class);
    job.setJarByClass(KMeansMapper.class);

    FileInputFormat.addInputPath(job, in);
    FileSystem fs = FileSystem.get(conf);
    if (fs.exists(out)) {
        fs.delete(out, true);
    }

    if (fs.exists(center)) {
        fs.delete(out, true);
    }

    if (fs.exists(in)) {
        fs.delete(in, true);
    }

    writeExampleCenters(conf, center, fs);

    writeExampleVectors(conf, in, fs);

    FileOutputFormat.setOutputPath(job, out);
    job.setInputFormatClass(SequenceFileInputFormat.class);
    job.setOutputFormatClass(SequenceFileOutputFormat.class);

    job.setOutputKeyClass(ClusterCenter.class);
    job.setOutputValueClass(VectorWritable.class);
```

centroid.path will hold the path to the current set of cluster centers

KMeansClusteringJob

```
public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {

    int iteration = 1;
    Configuration conf = new Configuration();

    conf.set("num.iteration", iteration + "");


    Path in = new Path("files/clustering/import/data");
    Path center = new Path("files/clustering /import/center/cen.seq");
    conf.set("centroid.path", center.toString());

    Path out = new Path("files/clustering/depth_1");


    Job job = Job.getInstance(conf);
    job.setJobName("KMeans Clustering");

    job.setMapperClass(KMeansMapper.class);
    job.setReducerClass(KMeansReducer.class);
    job.setJarByClass(KMeansMapper.class);

    FileInputFormat.addInputPath(job, in);
    FileSystem fs = FileSystem.get(conf);
    if (fs.exists(out)) {
        fs.delete(out, true);
    }

    if (fs.exists(center)) {
        fs.delete(out, true);
    }

    if (fs.exists(in)) {
        fs.delete(in, true);
    }

    writeExampleCenters(conf, center, fs);

    writeExampleVectors(conf, in, fs);

    FileOutputFormat.setOutputPath(job, out);
    job.setInputFormatClass(SequenceFileInputFormat.class);
    job.setOutputFormatClass(SequenceFileOutputFormat.class);

    job.setOutputKeyClass(ClusterCenter.class);
    job.setOutputValueClass(VectorWritable.class);
```

All of the job's configuration parameters are accessible to the mapper/reducer

KMeansClusteringJob

```java
public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {

    int iteration = 1;
    Configuration conf = new Configuration();

    conf.set("num.iteration", iteration + "");


    Path in = new Path("files/clustering/import/data");
    Path center = new Path("files/clustering /import/center/cen.seq");
    conf.set("centroid.path", center.toString());

    Path out = new Path("files/clustering/depth_1");


    Job job = Job.getInstance(conf);
    job.setJobName("KMeans Clustering");

    job.setMapperClass(KMeansMapper.class);
    job.setReducerClass(KMeansReducer.class);
    job.setJarByClass(KMeansMapper.class);

    FileInputFormat.addInputPath(job, in);
    FileSystem fs = FileSystem.get(conf);
    if (fs.exists(out)) {
        fs.delete(out, true);
    }

    if (fs.exists(center)) {
        fs.delete(out, true);
    }

    if (fs.exists(in)) {
        fs.delete(in, true);
    }

    writeExampleCenters(conf, center, fs);

    writeExampleVectors(conf, in, fs);

    FileOutputFormat.setOutputPath(job, out);
    job.setInputFormatClass(SequenceFileInputFormat.class);
    job.setOutputFormatClass(SequenceFileOutputFormat.class);

    job.setOutputKeyClass(ClusterCenter.class);
    job.setOutputValueClass(VectorWritable.class);
```

The mapper will use this path to read the current set of Cluster Centers

KMeansClusteringJob

```java
public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {

    int iteration = 1;
    Configuration conf = new Configuration();

    conf.set("num.iteration", iteration + "");


    Path in = new Path("files/clustering/import/data");
    Path center = new Path("files/clustering /import/center/cen.seq");
    conf.set("centroid.path", center.toString());
    Path out = new Path("files/clustering/depth_1");

    Job job = Job.getInstance(conf);
    job.setJobName("KMeans Clustering");

    job.setMapperClass(KMeansMapper.class);
    job.setReducerClass(KMeansReducer.class);
    job.setJarByClass(KMeansMapper.class);

    FileInputFormat.addInputPath(job, in);
    FileSystem fs = FileSystem.get(conf);
    if (fs.exists(out)) {
        fs.delete(out, true);
    }

    if (fs.exists(center)) {
        fs.delete(out, true);
    }

    if (fs.exists(in)) {
        fs.delete(in, true);
    }

    writeExampleCenters(conf, center, fs);

    writeExampleVectors(conf, in, fs);

    FileOutputFormat.setOutputPath(job, out);
    job.setInputFormatClass(SequenceFileInputFormat.class);
    job.setOutputFormatClass(SequenceFileOutputFormat.class);

    job.setOutputKeyClass(ClusterCenter.class);
    job.setOutputValueClass(VectorWritable.class);
```

We want to start the first iteration

KMeansClusteringJob

```java
public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {

    int iteration = 1;
    Configuration conf = new Configuration();

    conf.set("num.iteration", iteration + "");


    Path in = new Path("files/clustering/import/data");
    Path center = new Path("files/clustering /import/center/cen.seq");
    conf.set("centroid.path", center.toString());
    Path out = new Path("files/clustering/depth_1");

    Job job = Job.getInstance(conf);
    job.setJobName("KMeans Clustering");

    job.setMapperClass(KMeansMapper.class);
    job.setReducerClass(KMeansReducer.class);
    job.setJarByClass(KMeansMapper.class);

    FileInputFormat.addInputPath(job, in);
    FileSystem fs = FileSystem.get(conf);
    if (fs.exists(out)) {
        fs.delete(out, true);
    }

    if (fs.exists(center)) {
        fs.delete(out, true);
    }

    if (fs.exists(in)) {
        fs.delete(in, true);
    }

    writeExampleCenters(conf, center, fs);

    writeExampleVectors(conf, in, fs);

    FileOutputFormat.setOutputPath(job, out);
    job.setInputFormatClass(SequenceFileInputFormat.class);
    job.setOutputFormatClass(SequenceFileOutputFormat.class);
```

The output will be written to the path specified here

KMeansClusteringJob

```java
public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {

    int iteration = 1;
    Configuration conf = new Configuration();
    conf.set("num.iteration", iteration + "");

    Path in = new Path("files/clustering/import/data");
    Path center = new Path("files/clustering /import/center/cen.seq");
    conf.set("centroid.path", center.toString());
    Path out = new Path("files/clustering/depth_1");

    Job job = Job.getInstance(conf);
    job.setJobName("KMeans Clustering");

    job.setMapperClass(KMeansMapper.class);
    job.setReducerClass(KMeansReducer.class);
    job.setJarByClass(KMeansMapper.class);

    FileInputFormat.addInputPath(job, in);
    FileSystem fs = FileSystem.get(conf);
    if (fs.exists(out)) {
     fs.delete(out, true);
    }

    if (fs.exists(center)) {
     fs.delete(out, true);
    }

    if (fs.exists(in)) {
     fs.delete(in, true);
    }
    writeExampleCenters(conf, center, fs);
    writeExampleVectors(conf, in, fs);
```

Before starting, we clean up the paths where we want to store data

KMeansClusteringJob

```java
Path out = new Path("files/clustering/depth_1");


Job job = Job.getInstance(conf);
job.setJobName("KMeans Clustering");

job.setMapperClass(KMeansMapper.class);
job.setReducerClass(KMeansReducer.class);
job.setJarByClass(KMeansMapper.class);

FileInputFormat.addInputPath(job, in);
FileSystem fs = FileSystem.get(conf);
if (fs.exists(out)) {
      fs.delete(out, true);
}

if (fs.exists(center)) {
      fs.delete(out, true);
}

if (fs.exists(in)) {
      fs.delete(in, true);
}

writeExampleCenters(conf, center, fs);


writeExampleVectors(conf, in, fs);

FileOutputFormat.setOutputPath(job, out);
job.setInputFormatClass(SequenceFileInputFormat.class);
job.setOutputFormatClass(SequenceFileOutputFormat.class);

job.setOutputKeyClass(ClusterCenter.class);
job.setOutputValueClass(VectorWritable.class);

job.waitForCompletion(true);

long counter = job.getCounters().findCounter(KMeansReducer.Counter.CONVERGED).getValue();
iteration++;
while (counter > 0) {
      conf = new Configuration();
      conf.set("centroid.path", center.toString());
      conf.set("num.iteration", iteration + "");
      job = Job.getInstance(conf);
      job.setJobName("KMeans Clustering " + iteration);

      job.setMapperClass(KMeansMapper.class);
      job.setReducerClass(KMeansReducer.class);
      job.setJarByClass(KMeansMapper.class);

      in = new Path("files/clustering/depth_" + (iteration - 1) + "/");
      out = new Path("files/clustering/depth_" + iteration);

      FileInputFormat.addInputPath(job, in);
      if (fs.exists(out))
            fs.delete(out, true);

      FileOutputFormat.setOutputPath(job, out);
      job.setInputFormatClass(SequenceFileInputFormat.class);
      job.setOutputFormatClass(SequenceFileOutputFormat.class);
      job.setOutputKeyClass(ClusterCenter.class);
      job.setOutputValueClass(VectorWritable.class);

      job.waitForCompletion(true);
      iteration++;
      counter = job.getCounters().findCounter(KMeansReducer.Counter.CONVERGED).getValue();
}

Path result = new Path("files/clustering/depth_" + (iteration - 1) + "/");

 FileStatus[] stati = fs.listStatus(result);
 for (FileStatus status : stati) {
     if (!status.isDirectory()) {
         Path path = status.getPath();
         if (!path.getName().equals("_SUCCESS")) {
             //LOG.info("FOUND " + path.toString());
             try (SequenceFile.Reader reader = new SequenceFile.Reader(fs, path, conf)) {

                 ClusterCenter key = new ClusterCenter();
                 VectorWritable v = new VectorWritable();
                 while (reader.next(key, v)) {
                    // LOG.info(key + " / " + v);
                     System.out.println( key + " /" + v);

                 }
             }
         }
     }
 }
```

These are 2 helper methods that will write some sample data to be processed

```
                                      VectorWritable v = new VectorWritable();
                                      while (reader.next(key, v)) {
                                         // LOG.info(key + " / " + v);
                                          System.out.println( key + " /" + v);

                                      }
                                 }
                           }
                      }
                }
      }
```

KMeansClusteringJob

```
public static void writeExampleVectors(Configuration conf, Path in,
FileSystem fs) throws IOException {
  try (SequenceFile.Writer dataWriter = SequenceFile.createWriter(fs, conf, in, ClusterCenter.class,
      VectorWritable.class)) {
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(1, 2));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(16, 3));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(3, 3));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(2, 2));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(2, 3));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(25, 1));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(7, 6));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(6, 5));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(-1, -23));
  }
}
```

This writes some example input to a **Sequence File**

```
            ClusterCenter key = new ClusterCenter();
            VectorWritable v = new VectorWritable();
            while (reader.next(key, v)) {
               // LOG.info(key + " / " + v);
                System.out.println( key + " /" + v);

            }
          }
        }
      }
    }
}
```

KMeansClusteringJob

```
public static void writeExampleVectors(Configuration conf, Path in, FileSystem fs)
throws IOException {
  try (SequenceFile.Writer dataWriter =
SequenceFile.createWriter(fs, conf, in,
ClusterCenter.class,VectorWritable.class)) {
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(1, 2));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(16, 3));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(3, 3));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(2, 2));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(2, 3));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(25, 1));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(7, 6));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(6, 5));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(-1, -23));
  }
}
```

A sequence file stores key, value pairs in raw binary form

```
                          while (reader.next(key, v)) {
                            // LOG.info(key + " / " + v);
                             System.out.println( key + " /" + v);

                          }
                      }
                  }
              }
          }
  }
```

KMeansClusteringJob

```
public static void writeExampleVectors(Configuration conf, Path in,
FileSystem fs) throws IOException {
  try (SequenceFile.Writer dataWriter = SequenceFile.createWriter(fs, conf, in,
ClusterCenter.class,VectorWritable.class)) {
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(1, 2));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(16, 3));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(3, 3));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(2, 2));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(2, 3));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(25, 1));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(7, 6));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(6, 5));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(-1, -23));
  }
}
```

Given the types of the key and value, the Sequence file can be deserialized into the specified object types

```
              while (reader.next(key, v)) {
                // LOG.info(key + " / " + v);
                 System.out.println( key + " /" + v);

              }
            }
          }
        }
      }
  }
```

KMeansClusteringJob

```
public static void writeExampleVectors(Configuration conf, Path in,
FileSystem fs) throws IOException {
  try (SequenceFile.Writer dataWriter = SequenceFile.createWriter(fs, conf, in,
ClusterCenter.class,VectorWritable.class)) {
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(1, 2));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(16, 3));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(3, 3));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(2, 2));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(2, 3));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(25, 1));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(7, 6));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(6, 5));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(-1, -23));
  }
}
```

If instead our data had been stored as text files, we would need to parse the text and construct the Vectors ourselves

KMeansClusteringJob

```
                //LOG.info("FOUND " + path.toString());
                try (SequenceFile.Reader reader = new SequenceFile.Reader(fs, path, conf)) {

                    ClusterCenter key = new ClusterCenter();
                    VectorWritable v = new VectorWritable();
                    while (reader.next(key, v)) {
                        // LOG.info(key + " / " + v);
                         System.out.println( key + " /" + v);

                    }
                }
            }
        }
    }
}

public static void writeExampleVectors(Configuration conf, Path in, FileSystem fs) throws IOException {
     try (SequenceFile.Writer dataWriter = SequenceFile.createWriter(fs, conf, in, ClusterCenter.class,VectorWritable.class)) {
dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(1, 2));
        dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(16, 3));
        dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(3, 3));
        dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(2, 2));
        dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(2, 3));
        dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(25, 1));
        dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(7, 6));
        dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(6, 5));
        dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(-1, -23));
    }
}
```

Each row is a key value pair of Cluster Center and a Data point

KMeansClusteringJob

```
            //LOG.info("FOUND " + path.toString());
            try (SequenceFile.Reader reader = new SequenceFile.Reader(fs, path, conf)) {

                ClusterCenter key = new ClusterCenter();
                VectorWritable v = new VectorWritable();
                while (reader.next(key, v)) {
                    // LOG.info(key + " / " + v);
                     System.out.println( key + " /" + v);
                }
            }
        }
    }
}

public static void writeExampleVectors(Configuration conf, Path in, FileSystem fs) throws IOException {
     try (SequenceFile.Writer dataWriter = SequenceFile.createWriter(fs, conf, in, ClusterCenter.class,VectorWritable.class)) {
dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(1, 2));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(16, 3));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(3, 3));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(2, 2));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(2, 3));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(25, 1));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(7, 6));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(6, 5));
    dataWriter.append(new ClusterCenter(new VectorWritable(0, 0)), new VectorWritable(-1, -23));
  }
}
```

The Cluster Centers have been initialized to the origin for all the data points

KMeansClusteringJob

We specify that the input is in the form of a Sequence file

```
Path center = new Path("files/clustering/import/center/cen.seq");
conf.set("centroid.path", center.toString());

Path out = new Path("files/clustering/depth_1");

Job job = Job.getInstance(conf);
job.setJobName("KMeans Clustering");

job.setMapperClass(KMeansMapper.class);
job.setReducerClass(KMeansReducer.class);
job.setJarByClass(KMeansMapper.class);

FileInputFormat.addInputPath(job, in);
FileSystem fs = FileSystem.get(conf);
if (fs.exists(out)) {
    fs.delete(out, true);
}

if (fs.exists(center)) {
    fs.delete(out, true);
}

if (fs.exists(in)) {
    fs.delete(in, true);
}

writeExampleCenters(conf, center, fs);

writeExampleVectors(conf, in, fs);

FileOutputFormat.setOutputPath(job, out);
```

```
job.setInputFormatClass(SequenceFileInputFormat.class)
```

```
job.setOutputFormatClass(SequenceFileOutputFormat.class);

job.setOutputKeyClass(ClusterCenter.class);
job.setOutputValueClass(VectorWritable.class);

job.waitForCompletion(true);

long counter = job.getCounters().findCounter(KMeansReducer.Counter.CONVERGED).getValue();
iteration++;
while (counter > 0) {
    conf = new Configuration();
    conf.set("centroid.path", center.toString());
    conf.set("num.iteration", iteration + "");
    job = Job.getInstance(conf);
    job.setJobName("KMeans Clustering " + iteration);

    job.setMapperClass(KMeansMapper.class);
    job.setReducerClass(KMeansReducer.class);
    job.setJarByClass(KMeansMapper.class);

    in = new Path("files/clustering/depth_" + (iteration - 1) + "/");
    out = new Path("files/clustering/depth_" + iteration);

    FileInputFormat.addInputPath(job, in);
    if (fs.exists(out))
        fs.delete(out, true);

    FileOutputFormat.setOutputPath(job, out);
    job.setInputFormatClass(SequenceFileInputFormat.class);
    job.setOutputFormatClass(SequenceFileOutputFormat.class);
```

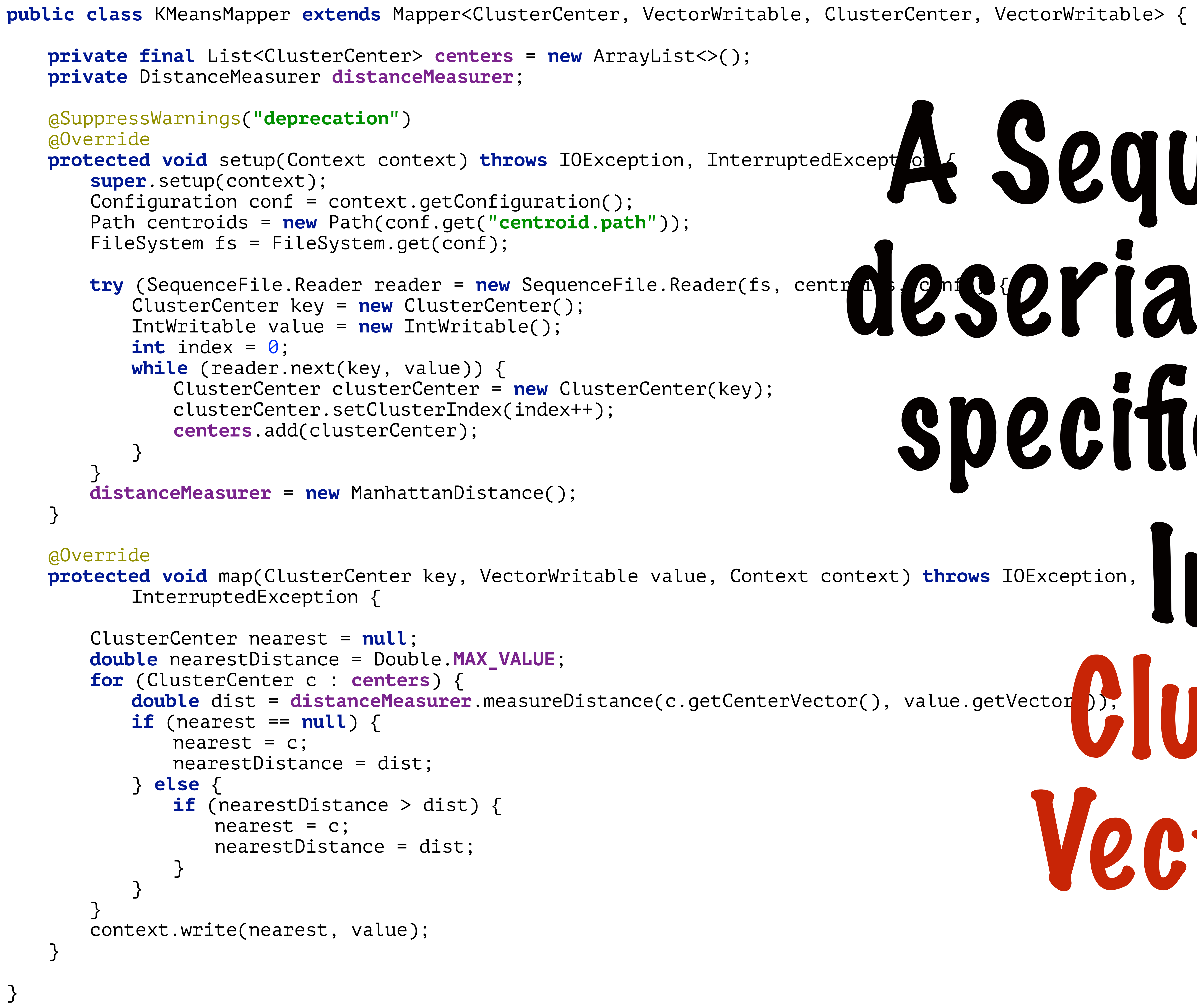
KMeansClusteringJob

```
job.setReducerClass(KMeansReducer.class);
job.setJarByClass(KMeansMapper.class);

FileInputFormat.addInputPath(job, in);
FileSystem fs = FileSystem.get(conf);
if (fs.exists(out)) {
    fs.delete(out, true);
}

if (fs.exists(center)) {
    fs.delete(out, true);
}

if (fs.exists(in)) {
    fs.delete(in, true);
}

writeExampleCenters(conf, center, fs);

writeExampleVectors(conf, in, fs);

FileOutputFormat.setOutputPath(job, out);
job.setInputFormatClass(SequenceFileInputFormat.class);
job.setOutputFormatClass(SequenceFileOutputFormat.class);

job.setOutputKeyClass(ClusterCenter.class);
job.setOutputValueClass(VectorWritable.class);

job.waitForCompletion(true);

long counter = job.getCounters().findCounter(KMeansReducer.Counter.CONVERGED).getValue();
iteration++;
while (counter > 0) {
    conf = new Configuration();
    conf.set("centroid.path", center.toString());
    conf.set("num.iteration", iteration + "");
    job = Job.getInstance(conf);
    job.setJobName("KMeans Clustering " + iteration);

    job.setMapperClass(KMeansMapper.class);
    job.setReducerClass(KMeansReducer.class);
    job.setJarByClass(KMeansMapper.class);

    in = new Path("files/clustering/depth_" + (iteration - 1) + "/");
    out = new Path("files/clustering/depth_" + iteration);

    FileInputFormat.addInputPath(job, in);
    if (fs.exists(out))
        fs.delete(out, true);

    FileOutputFormat.setOutputPath(job, out);
    job.setInputFormatClass(SequenceFileInputFormat.class);
    job.setOutputFormatClass(SequenceFileOutputFormat.class);
    job.setOutputKeyClass(ClusterCenter.class);
    job.setOutputValueClass(VectorWritable.class);

    job.waitForCompletion(true);
    iteration++;
    counter = job.getCounters().findCounter(KMeansReducer.Counter.CONVERGED).getValue();
}
```

Finally we start the first iteration

Let’s now step into the Mapper

KMeansMapper



```java
public class KMeansMapper extends Mapper<ClusterCenter, VectorWritable, ClusterCenter, VectorWritable> {

    private final List<ClusterCenter> centers = new ArrayList<>();
    private DistanceMeasurer distanceMeasurer;

    @SuppressWarnings("deprecation")
    @Override
    protected void setup(Context context) throws IOException, InterruptedException {
        super.setup(context);
        Configuration conf = context.getConfiguration();
        Path centroids = new Path(conf.get("centroid.path"));
        FileSystem fs = FileSystem.get(conf);

        try (SequenceFile.Reader reader = new SequenceFile.Reader(fs, centroids, conf)) {
            ClusterCenter key = new ClusterCenter();
            IntWritable value = new IntWritable();
            int index = 0;
            while (reader.next(key, value)) {
                ClusterCenter clusterCenter = new ClusterCenter(key);
                clusterCenter.setClusterIndex(index++);
                centers.add(clusterCenter);
            }
        }
        distanceMeasurer = new ManhattanDistance();
    }

    @Override
    protected void map(ClusterCenter key, VectorWritable value, Context context) throws IOException,
            InterruptedException {

        ClusterCenter nearest = null;
        double nearestDistance = Double.MAX_VALUE;
        for (ClusterCenter c : centers) {
            double dist = distanceMeasurer.measureDistance(c.getCenterVector(), value.getVector());
            if (nearest == null) {
                nearest = c;
                nearestDistance = dist;
            } else {
                if (nearestDistance > dist) {
                    nearest = c;
                    nearestDistance = dist;
                }
            }
        }
        context.write(nearest, value);
    }

}
```

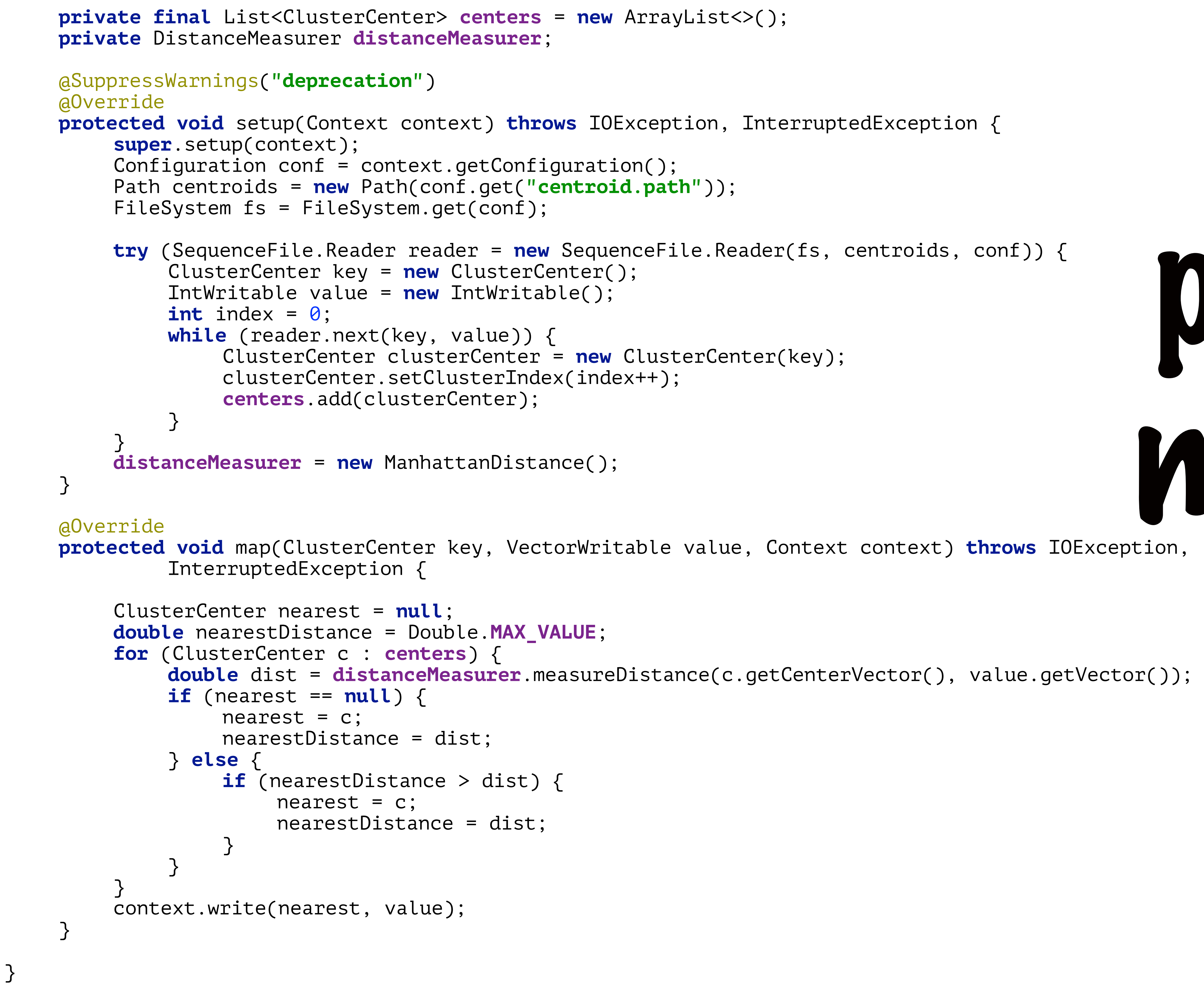
A SequenceFile can be deserialized directly to specified object types

In our case

ClusterCenter, VectorWritable

KMeansMapper

```java
public class KMeansMapper extends
Mapper<ClusterCenter, VectorWritable,
ClusterCenter, VectorWritable> {
```

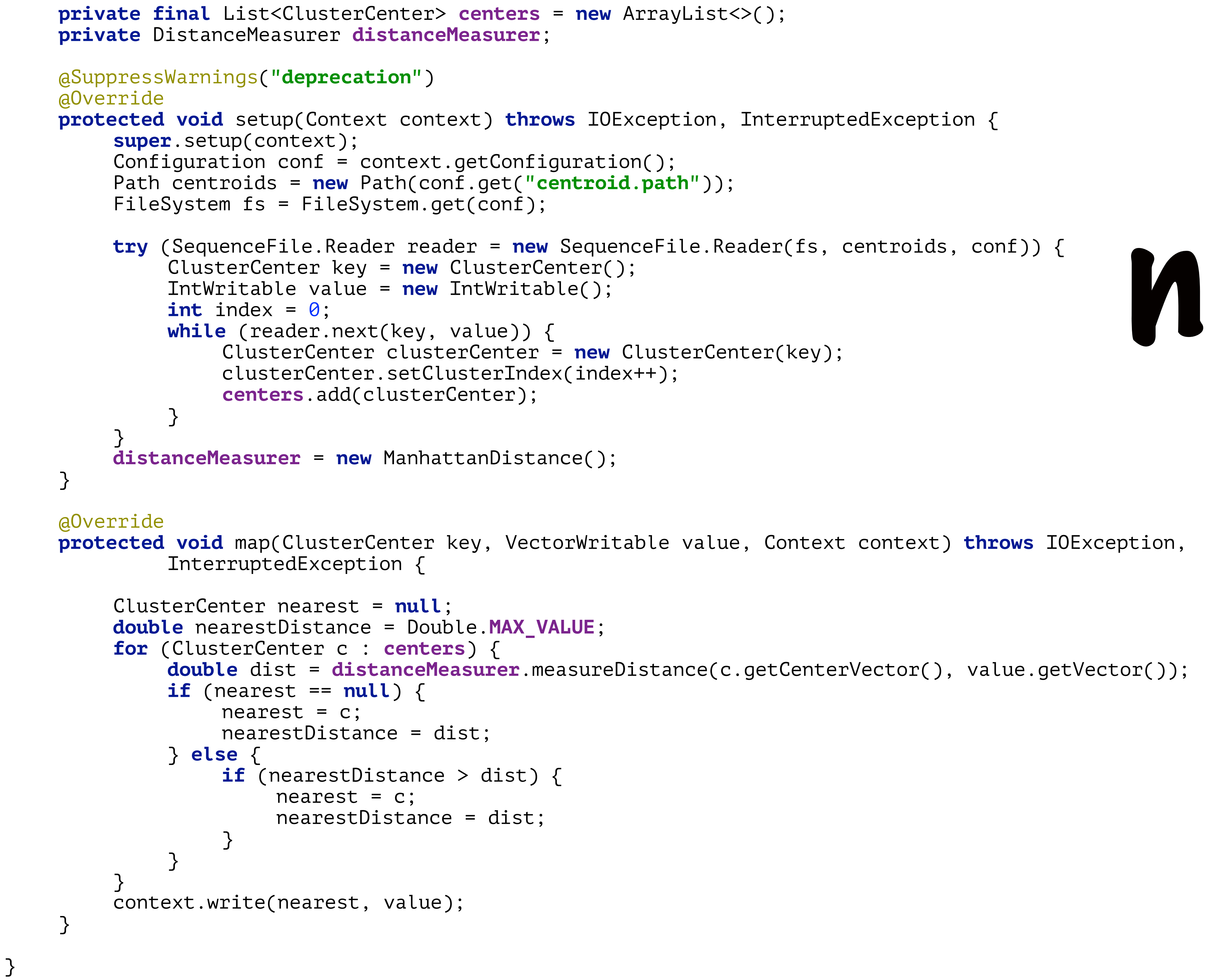


```java
    private final List<ClusterCenter> centers = new ArrayList<>();
    private DistanceMeasurer distanceMeasurer;

    @SuppressWarnings("deprecation")
    @Override
    protected void setup(Context context) throws IOException, InterruptedException {
        super.setup(context);
        Configuration conf = context.getConfiguration();
        Path centroids = new Path(conf.get("centroid.path"));
        FileSystem fs = FileSystem.get(conf);

        try (SequenceFile.Reader reader = new SequenceFile.Reader(fs, centroids, conf)) {
            ClusterCenter key = new ClusterCenter();
            IntWritable value = new IntWritable();
            int index = 0;
            while (reader.next(key, value)) {
                ClusterCenter clusterCenter = new ClusterCenter(key);
                clusterCenter.setClusterIndex(index++);
                centers.add(clusterCenter);
            }
        }
        distanceMeasurer = new ManhattanDistance();
    }

    @Override
    protected void map(ClusterCenter key, VectorWritable value, Context context) throws IOException,
            InterruptedException {

        ClusterCenter nearest = null;
        double nearestDistance = Double.MAX_VALUE;
        for (ClusterCenter c : centers) {
            double dist = distanceMeasurer.measureDistance(c.getCenterVector(), value.getVector());
            if (nearest == null) {
                nearest = c;
                nearestDistance = dist;
            } else {
                if (nearestDistance > dist) {
                    nearest = c;
                    nearestDistance = dist;
                }
            }
        }
        context.write(nearest, value);
    }

}
```

Each record is a data point with its current nearest Cluster Center

KMeansMapper

```
public class KMeansMapper extends
Mapper<ClusterCenter, VectorWritable,
ClusterCenter, VectorWritable> {
```



```
    private final List<ClusterCenter> centers = new ArrayList<>();
    private DistanceMeasurer distanceMeasurer;

    @SuppressWarnings("deprecation")
    @Override
    protected void setup(Context context) throws IOException, InterruptedException {
        super.setup(context);
        Configuration conf = context.getConfiguration();
        Path centroids = new Path(conf.get("centroid.path"));
        FileSystem fs = FileSystem.get(conf);

        try (SequenceFile.Reader reader = new SequenceFile.Reader(fs, centroids, conf)) {
            ClusterCenter key = new ClusterCenter();
            IntWritable value = new IntWritable();
            int index = 0;
            while (reader.next(key, value)) {
                ClusterCenter clusterCenter = new ClusterCenter(key);
                clusterCenter.setClusterIndex(index++);
                centers.add(clusterCenter);
            }
        }
        distanceMeasurer = new ManhattanDistance();
    }

    @Override
    protected void map(ClusterCenter key, VectorWritable value, Context context) throws IOException,
            InterruptedException {

        ClusterCenter nearest = null;
        double nearestDistance = Double.MAX_VALUE;
        for (ClusterCenter c : centers) {
            double dist = distanceMeasurer.measureDistance(c.getCenterVector(), value.getVector());
            if (nearest == null) {
                nearest = c;
                nearestDistance = dist;
            } else {
                if (nearestDistance > dist) {
                    nearest = c;
                    nearestDistance = dist;
                }
            }
        }
        context.write(nearest, value);
    }

}
```

The output will be nearest Cluster Center, the data point

KMeansMapper

```java
public class KMeansMapper extends Mapper<ClusterCenter, VectorWritable, ClusterCenter, VectorWritable> {

    private final List<ClusterCenter> centers = new ArrayList<>();
    private DistanceMeasurer distanceMeasurer;

    @SuppressWarnings("deprecation")
    @Override
    protected void setup(Context context) throws IOException, InterruptedException {
        super.setup(context);
        Configuration conf = context.getConfiguration();
        Path centroids = new Path(conf.get("centroid.path"));
        FileSystem fs = FileSystem.get(conf);

        try (SequenceFile.Reader reader = new SequenceFile.Reader(fs, centroids, conf)) {
            ClusterCenter key = new ClusterCenter();
            IntWritable value = new IntWritable();
            int index = 0;
            while (reader.next(key, value)) {
                ClusterCenter clusterCenter = new ClusterCenter(key);
                clusterCenter.setClusterIndex(index++);
                centers.add(clusterCenter);
            }
        }
        distanceMeasurer = new ManhattanDistance();
    }

    @Override
    protected void map(ClusterCenter key, VectorWritable value, Context context) throws IOException,
            InterruptedException {

        ClusterCenter nearest = null;
        double nearestDistance = Double.MAX_VALUE;
        for (ClusterCenter c : centers) {
            double dist = distanceMeasurer.measureDistance(c.getCenterVector(), value.getVector());
            if (nearest == null) {
                nearest = c;
                nearestDistance = dist;
            } else {
                if (nearestDistance > dist) {
                    nearest = c;
                    nearestDistance = dist;
                }
            }
        }
        context.write(nearest, value);
    }

}
```

The nearest Cluster center is calculated by iterating through a list of Cluster Centers

KMeansMapper

```java
public class KMeansMapper extends Mapper<ClusterCenter, VectorWritable, ClusterCenter, VectorWritable> {


    private final List<ClusterCenter> centers = new ArrayList<>();
    private DistanceMeasurer distanceMeasurer;

    @SuppressWarnings("deprecation")
    @Override
    protected void setup(Context context) throws IOException, InterruptedException {
        super.setup(context);
        Configuration conf = context.getConfiguration();
        Path centroids = new Path(conf.get("centroid.path"));
        FileSystem fs = FileSystem.get(conf);

        try (SequenceFile.Reader reader = new SequenceFile.Reader(fs, centroids, conf)) {
            ClusterCenter key = new ClusterCenter();
            IntWritable value = new IntWritable();
            int index = 0;
            while (reader.next(key, value)) {
                ClusterCenter clusterCenter = new ClusterCenter(key);
                clusterCenter.setClusterIndex(index++);
                centers.add(clusterCenter);
            }
        }
        distanceMeasurer = new ManhattanDistance();
    }

    @Override
    protected void map(ClusterCenter key, VectorWritable value, Context context) throws IOException,
            InterruptedException {

        ClusterCenter nearest = null;
        double nearestDistance = Double.MAX_VALUE;
        for (ClusterCenter c : centers) {
            double dist = distanceMeasurer.measureDistance(c.getCenterVector(), value.getVector());
            if (nearest == null) {
                nearest = c;
                nearestDistance = dist;
            } else {
                if (nearestDistance > dist) {
                    nearest = c;
                    nearestDistance = dist;
                }
            }
        }
        context.write(nearest, value);
    }

}
```

The Cluster Centers are always stored in a file whose path can be accessed using the Configuration

KMeansMapper

```
public class KMeansMapper extends Mapper<ClusterCenter, VectorWritable, ClusterCenter, VectorWritable> {

    private final List<ClusterCenter> centers = new ArrayList<>();
    private DistanceMeasurer distanceMeasurer;

    @SuppressWarnings("deprecation")
    @Override
    protected void setup(Context context) throws IOException, InterruptedException {
        super.setup(context);
        Configuration conf = context.getConfiguration();

        Path centroids = new Path(conf.get("centroid.path"));
        FileSystem fs = FileSystem.get(conf);

        try (SequenceFile.Reader reader = new SequenceFile.Reader(fs,
centroids, conf)) {
            ClusterCenter key = new ClusterCenter();
            IntWritable value = new IntWritable();
            int index = 0;
            while (reader.next(key, value)) {
              ClusterCenter clusterCenter = new ClusterCenter(key);
              clusterCenter.setClusterIndex(index++);
              centers.add(clusterCenter);
            }
        }
        distanceMeasurer = new ManhattanDistance();
    }

    @Override
    protected void map(ClusterCenter key, VectorWritable value, Context context) throws IOException,
            InterruptedException {

        ClusterCenter nearest = null;
        double nearestDistance = Double.MAX_VALUE;
        for (ClusterCenter c : centers) {
            double dist = distanceMeasurer.measureDistance(c.getCenterVector(), value.getVector());
            if (nearest == null) {
                nearest = c;
                nearestDistance = dist;
            } else {
                if (nearestDistance > dist) {
                    nearest = c;
                    nearestDistance = dist;
                }
            }
        }
        context.write(nearest, value);
    }

}
```

We read the Sequence File and add the data to a list of Cluster Centers

KMeansMapper

```
public class KMeansMapper extends Mapper<ClusterCenter, VectorWritable, ClusterCenter, VectorWritable> {

    private final List<ClusterCenter> centers = new ArrayList<>();
    private DistanceMeasurer distanceMeasurer;

    @SuppressWarnings("deprecation")
    @Override
    protected void setup(Context context) throws IOException,
InterruptedException {
        super.setup(context);
        Configuration conf = context.getConfiguration();

        Path centroids = new Path(conf.get("centroid.path"));
        FileSystem fs = FileSystem.get(conf);

        try (SequenceFile.Reader reader = new SequenceFile.Reader(fs, centroids, conf)) {
            ClusterCenter key = new ClusterCenter();
            IntWritable value = new IntWritable();
            int index = 0;
            while (reader.next(key, value)) {
                ClusterCenter clusterCenter = new ClusterCenter(key);
                clusterCenter.setClusterIndex(index++);
                centers.add(clusterCenter);
            }


        }
        distanceMeasurer = new ManhattanDistance();
    }

    @Override
    protected void map(ClusterCenter key, VectorWritable value, Context context) throws IOException,
            InterruptedException {

        ClusterCenter nearest = null;
        double nearestDistance = Double.MAX_VALUE;
        for (ClusterCenter c : centers) {
            double dist = distanceMeasurer.measureDistance(c.getCenterVector(), value.getVector());
            if (nearest == null) {
                nearest = c;
                nearestDistance = dist;
            } else {
                if (nearestDistance > dist) {
                    nearest = c;
                    nearestDistance = dist;
                }
            }
        }
        context.write(nearest, value);
    }

}
```

This read is done in the setup() method of the Mapper, which is called before any processing starts

KMeansMapper

```java
public class KMeansMapper extends Mapper<ClusterCenter, VectorWritable, ClusterCenter, VectorWritable> {

    private final List<ClusterCenter> centers = new ArrayList<>();
    private DistanceMeasurer distanceMeasurer;

    @SuppressWarnings("deprecation")
    @Override
    protected void setup(Context context) throws IOException,
InterruptedException {
        super.setup(context);
        Configuration conf = context.getConfiguration();

        Path centroids = new Path(conf.get("centroid.path"));
        FileSystem fs = FileSystem.get(conf);

        try (SequenceFile.Reader reader = new SequenceFile.Reader(fs, centroids, conf)) {
            ClusterCenter key = new ClusterCenter();
            IntWritable value = new IntWritable();
            int index = 0;
            while (reader.next(key, value)) {
                ClusterCenter clusterCenter = new ClusterCenter(key);
                clusterCenter.setClusterIndex(index++);
                centers.add(clusterCenter);
            }


        }
        distanceMeasurer = new ManhattanDistance();
    }

    @Override
    protected void map(ClusterCenter key, VectorWritable value, Context context) throws IOException,
            InterruptedException {

        ClusterCenter nearest = null;
        double nearestDistance = Double.MAX_VALUE;
        for (ClusterCenter c : centers) {
            double dist = distanceMeasurer.measureDistance(c.getCenterVector(), value.getVector());
            if (nearest == null) {
                nearest = c;
                nearestDistance = dist;
            } else {
                if (nearestDistance > dist) {
                    nearest = c;
                    nearestDistance = dist;
                }
            }
        }
        context.write(nearest, value);
    }

}
```

We also setup a DistanceMeasurer here

KMeansMapper

```
private final List<ClusterCenter> centers = new ArrayList<>();
private DistanceMeasurer distanceMeasurer;

@SuppressWarnings("deprecation")
@Override
protected void setup(Context context) throws IOException, InterruptedException {
      super.setup(context);
      Configuration conf = context.getConfiguration();
      Path centroids = new Path(conf.get("centroid.path"));
      FileSystem fs = FileSystem.get(conf);

      try (SequenceFile.Reader reader = new SequenceFile.Reader(fs, centroids, conf)) {
            ClusterCenter key = new ClusterCenter();
            IntWritable value = new IntWritable();
            int index = 0;
            while (reader.next(key, value)) {
                  ClusterCenter clusterCenter = new ClusterCenter(key);
                  clusterCenter.setClusterIndex(index++);
                  centers.add(clusterCenter);
            }
      }
      distanceMeasurer = new ManhattanDistance();
}

@Override
protected void map(ClusterCenter key, VectorWritable value, Context context) throws IOException,
            InterruptedException {

      ClusterCenter nearest = null;


      double nearestDistance = Double.MAX_VALUE;
```

```
  for (ClusterCenter c : centers) {
double dist = distanceMeasurer.measureDistance(c.getCenterVector(),
value.getVector());
    if (nearest == null) {
      nearest = c;
      nearestDistance = dist;
    } else {
      if (nearestDistance > dist) {
        nearest = c;
        nearestDistance = dist;
      }
    }
```

The map() method will simply iterate through the list of centers and find the nearest one

KMeansMapper

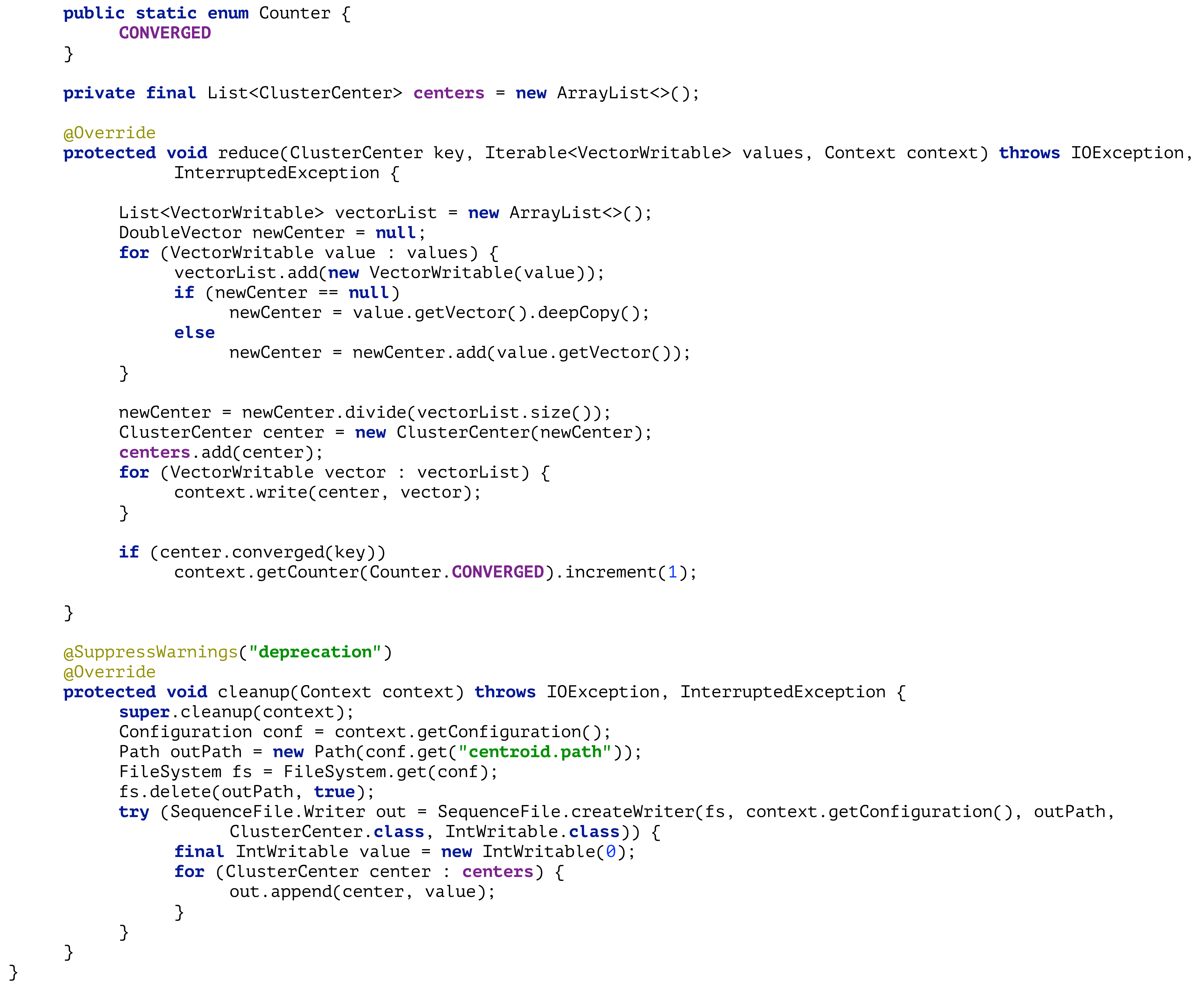
```java
    private final List<ClusterCenter> centers = new ArrayList<>();
    private DistanceMeasurer distanceMeasurer;

    @SuppressWarnings("deprecation")
    @Override
    protected void setup(Context context) throws IOException, InterruptedException {
        super.setup(context);
        Configuration conf = context.getConfiguration();
        Path centroids = new Path(conf.get("centroid.path"));
        FileSystem fs = FileSystem.get(conf);

        try (SequenceFile.Reader reader = new SequenceFile.Reader(fs, centroids, conf)) {
            ClusterCenter key = new ClusterCenter();
            IntWritable value = new IntWritable();
            int index = 0;
            while (reader.next(key, value)) {
                ClusterCenter clusterCenter = new ClusterCenter(key);
                clusterCenter.setClusterIndex(index++);
                centers.add(clusterCenter);
            }
        }
        distanceMeasurer = new ManhattanDistance();
    }

    @Override
  protected void map(ClusterCenter key,
VectorWritable value, Context context)
throws IOException,
            InterruptedException {

        ClusterCenter nearest = null;
        double nearestDistance = Double.MAX_VALUE;
        for (ClusterCenter c : centers) {
            double dist = distanceMeasurer.measureDistance(c.getCenterVector(), value.getVector());
            if (nearest == null) {
                nearest = c;
                nearestDistance = dist;
            } else {
                if (nearestDistance > dist) {
                    nearest = c;
                    nearestDistance = dist;
                }
            }
        }
    context.write(nearest, value);
  }

}
```

The map() method will output <nearest Center, Data Point>

Let's now step in to the Reducer

KMeansReducer

```java
public class KMeansReducer extends
Reducer<ClusterCenter, VectorWritable,
ClusterCenter, VectorWritable> {
```



```java
    public static enum Counter {
        CONVERGED
    }

    private final List<ClusterCenter> centers = new ArrayList<>();

    @Override
    protected void reduce(ClusterCenter key, Iterable<VectorWritable> values, Context context) throws IOException,
            InterruptedException {

        List<VectorWritable> vectorList = new ArrayList<>();
        DoubleVector newCenter = null;
        for (VectorWritable value : values) {
            vectorList.add(new VectorWritable(value));
            if (newCenter == null)
                newCenter = value.getVector().deepCopy();
            else
                newCenter = newCenter.add(value.getVector());
        }

        newCenter = newCenter.divide(vectorList.size());
        ClusterCenter center = new ClusterCenter(newCenter);
        centers.add(center);
        for (VectorWritable vector : vectorList) {
            context.write(center, vector);
        }

        if (center.converged(key))
            context.getCounter(Counter.CONVERGED).increment(1);

    }

    @SuppressWarnings("deprecation")
    @Override
    protected void cleanup(Context context) throws IOException, InterruptedException {
        super.cleanup(context);
        Configuration conf = context.getConfiguration();
        Path outPath = new Path(conf.get("centroid.path"));
        FileSystem fs = FileSystem.get(conf);
        fs.delete(outPath, true);
        try (SequenceFile.Writer out = SequenceFile.createWriter(fs, context.getConfiguration(), outPath,
                ClusterCenter.class, IntWritable.class)) {
            final IntWritable value = new IntWritable(0);
            for (ClusterCenter center : centers) {
                out.append(center, value);
            }
        }
    }
}
```

The Reducer will compute new Cluster Center for all the data points assigned to 1 cluster

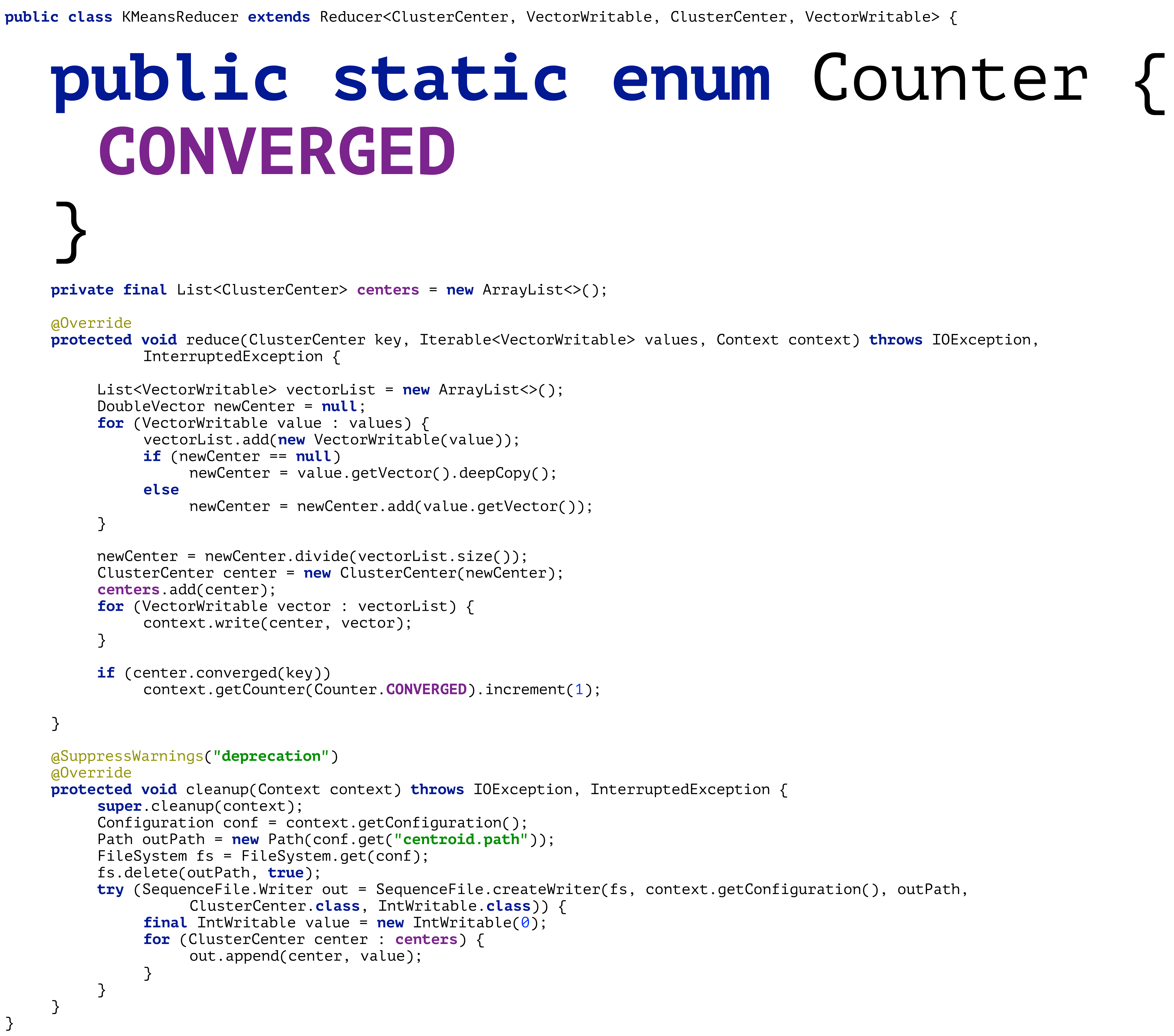
KMeansReducer

```
public class KMeansReducer extends
Reducer<ClusterCenter, VectorWritable,
ClusterCenter, VectorWritable> {

    public static enum Counter {
        CONVERGED
    }

    private final List<ClusterCenter> centers = new ArrayList<>();

    @Override
    protected void reduce(ClusterCenter key, Iterable<VectorWritable> values, Context context) throws IOException,
            InterruptedException {

        List<VectorWritable> vectorList = new ArrayList<>();
        DoubleVector newCenter = null;
        for (VectorWritable value : values) {
            vectorList.add(new VectorWritable(value));
            if (newCenter == null)
                newCenter = value.getVector().deepCopy();
            else
                newCenter = newCenter.add(value.getVector());
        }

        newCenter = newCenter.divide(vectorList.size());
        ClusterCenter center = new ClusterCenter(newCenter);
        centers.add(center);
        for (VectorWritable vector : vectorList) {
            context.write(center, vector);
        }

        if (center.converged(key))
            context.getCounter(Counter.CONVERGED).increment(1);

    }

    @SuppressWarnings("deprecation")
    @Override
    protected void cleanup(Context context) throws IOException, InterruptedException {
        super.cleanup(context);
        Configuration conf = context.getConfiguration();
        Path outPath = new Path(conf.get("centroid.path"));
        FileSystem fs = FileSystem.get(conf);
        fs.delete(outPath, true);
        try (SequenceFile.Writer out = SequenceFile.createWriter(fs, context.getConfiguration(), outPath,
                ClusterCenter.class, IntWritable.class)) {
            final IntWritable value = new IntWritable(0);
            for (ClusterCenter center : centers) {
                out.append(center, value);
            }
        }
    }
}
```

It will then check if new Cluster Center is different from old Cluster Center

If no, it increments a counter

KMeansReducer



```java
public class KMeansReducer extends Reducer<ClusterCenter, VectorWritable, ClusterCenter, VectorWritable> {

    public static enum Counter {
        CONVERGED
    }

    private final List<ClusterCenter> centers = new ArrayList<>();

    @Override
    protected void reduce(ClusterCenter key, Iterable<VectorWritable> values, Context context) throws IOException,
            InterruptedException {

        List<VectorWritable> vectorList = new ArrayList<>();
        DoubleVector newCenter = null;
        for (VectorWritable value : values) {
            vectorList.add(new VectorWritable(value));
            if (newCenter == null)
                newCenter = value.getVector().deepCopy();
            else
                newCenter = newCenter.add(value.getVector());
        }

        newCenter = newCenter.divide(vectorList.size());
        ClusterCenter center = new ClusterCenter(newCenter);
        centers.add(center);
        for (VectorWritable vector : vectorList) {
            context.write(center, vector);
        }

        if (center.converged(key))
            context.getCounter(Counter.CONVERGED).increment(1);

    }

    @SuppressWarnings("deprecation")
    @Override
    protected void cleanup(Context context) throws IOException, InterruptedException {
        super.cleanup(context);
        Configuration conf = context.getConfiguration();
        Path outPath = new Path(conf.get("centroid.path"));
        FileSystem fs = FileSystem.get(conf);
        fs.delete(outPath, true);
        try (SequenceFile.Writer out = SequenceFile.createWriter(fs, context.getConfiguration(), outPath,
                ClusterCenter.class, IntWritable.class)) {
            final IntWritable value = new IntWritable(0);
            for (ClusterCenter center : centers) {
                out.append(center, value);
            }
        }
    }
}
```

Here we set up that counter

KMeansReducer

```java
public class KMeansReducer extends Reducer<ClusterCenter, VectorWritable, ClusterCenter, VectorWritable> {

    public static enum Counter {
        CONVERGED
    }

    private final List<ClusterCenter> centers =
new ArrayList<>();

    @Override
    protected void reduce(ClusterCenter key, Iterable<VectorWritable> values, Context context) throws IOException,
            InterruptedException {

        List<VectorWritable> vectorList = new ArrayList<>();
        DoubleVector newCenter = null;
        for (VectorWritable value : values) {
            vectorList.add(new VectorWritable(value));
            if (newCenter == null)
                newCenter = value.getVector().deepCopy();
            else
                newCenter = newCenter.add(value.getVector());
        }

        newCenter = newCenter.divide(vectorList.size());
        ClusterCenter center = new ClusterCenter(newCenter);
        centers.add(center);
        for (VectorWritable vector : vectorList) {
            context.write(center, vector);
        }

        if (center.converged(key))
            context.getCounter(Counter.CONVERGED).increment(1);

    }

    @SuppressWarnings("deprecation")
    @Override
    protected void cleanup(Context context) throws IOException, InterruptedException {
        super.cleanup(context);
        Configuration conf = context.getConfiguration();
        Path outPath = new Path(conf.get("centroid.path"));
        FileSystem fs = FileSystem.get(conf);
        fs.delete(outPath, true);
        try (SequenceFile.Writer out = SequenceFile.createWriter(fs, context.getConfiguration(), outPath,
                ClusterCenter.class, IntWritable.class)) {
            final IntWritable value = new IntWritable(0);
            for (ClusterCenter center : centers) {
                out.append(center, value);
            }
        }
    }
}
```

We'll initialize a list to store new Cluster Centers

These will be written to a file at the end

KMeansReducer

```java
blic class KMeansReducer extends Reducer<ClusterCenter, VectorWritable, ClusterCenter, VectorWritable> {

    public static enum Counter {
        CONVERGED
    }

    private final List<ClusterCenter> centers = new ArrayList<>();

    @Override
    protected void reduce(ClusterCenter key, Iterable<VectorWritable> values, Context context) throws IOException,
            InterruptedException {

        List<VectorWritable> vectorList = new ArrayList<>();
        DoubleVector newCenter = null;
        for (VectorWritable value : values) {
            vectorList.add(new VectorWritable(value));
            if (newCenter == null)
                newCenter = value.getVector().deepCopy();
            else
                newCenter = newCenter.add(value.getVector());
        }

        newCenter = newCenter.divide(vectorList.size());
        ClusterCenter center = new ClusterCenter(newCenter);
        centers.add(center);
        for (VectorWritable vector : vectorList) {
            context.write(center, vector);
        }

        if (center.converged(key))
            context.getCounter(Counter.CONVERGED).increment(1);

    }

    @SuppressWarnings("deprecation")
    @Override
    protected void cleanup(Context context) throws IOException, InterruptedException {
        super.cleanup(context);
        Configuration conf = context.getConfiguration();
        Path outPath = new Path(conf.get("centroid.path"));
        FileSystem fs = FileSystem.get(conf);
        fs.delete(outPath, true);
        try (SequenceFile.Writer out = SequenceFile.createWriter(fs, context.getConfiguration(), outPath,
                    ClusterCenter.class, IntWritable.class)) {
            final IntWritable value = new IntWritable(0);
            for (ClusterCenter center : centers) {
                out.append(center, value);
            }
        }
    }
```

For each key (which represents 1 Cluster)

KMeansReducer

```
blic class KMeans[illegible]er exte[illegible]s [illegible]du[illegible]<C[illegible]ste[illegible]Ce[illegible]er, Ve[illegible]rit[illegible], [illegible]ust[illegible]Center[illegible]ectorWri[illegible]bl[illegible] {

    public stati[illegible] enu[illegible]unt[illegible] {
        CONV[illegible]ED
    }

    private final List<ClusterCenter> centers = new ArrayList<>();

    @Override
    protected void reduce(ClusterCenter key, Iterable<VectorWritable> values, Context context) throws IOException,
            InterruptedException {

        List<VectorWritable> vectorList = new ArrayList<>();
        DoubleVector newCenter = null;
        for (VectorWritable value : values) {
            vectorList.add(new VectorWritable(value));
            if (newCenter == null)
                newCenter = value.getVector().deepCopy();
            else
                newCenter = newCenter.add(value.getVector());
        }

        newCenter = newCenter.divide(vectorList.size());
        ClusterCenter center = new ClusterCenter(newCenter);
        centers.add(center);
        for (VectorWritable vector : vectorList) {
            context.write(center, vector);
        }

        if (center.converged(key))
            context.getCounter(Counter.CONVERGED).increment(1);

    }

    @SuppressWarnings("deprecation")
    @Override
    protected void cleanup(Context context) throws IOException, InterruptedException {
        super.cleanup(context);
        Configuration conf = context.getConfiguration();
        Path outPath = new Path(conf.get("centroid.path"));
        FileSystem fs = FileSystem.get(conf);
        fs.delete(outPath, true);
        try (SequenceFile.Writer out = SequenceFile.createWriter(fs, context.getConfiguration(), outPath,
                    ClusterCenter.class, IntWritable.class)) {
            final IntWritable value = new IntWritable(0);
            for (ClusterCenter center : centers) {
                out.append(center, value);
            }
        }
    }
```

For each key

We compute a new cluster center from the data points assigned to the cluster

KMeansReducer

```
ass KMeansReducer extends Reducer<ClusterCenter, VectorWritable, ClusterCenter, VectorWritable> {

ic static enum Counter {
 CONVERGED


ate final List<ClusterCenter> centers = new ArrayList<>();

rride
ected void reduce(ClusterCenter key, Iterable<VectorWritable> values, Context context) throws IOException,
       InterruptedException {

 List<VectorWritable> vectorList = new ArrayList<>();
 DoubleVector newCenter = null;
 for (VectorWritable value : values) {
       vectorList.add(new VectorWritable(value));
       if (newCenter == null)
             newCenter = value.getVector().deepCopy();
       else
             newCenter = newCenter.add(value.getVector());
 }

 newCenter = newCenter.divide(vectorList.size());
 ClusterCenter center = new ClusterCenter(newCenter);
 centers.add(center);
 for (VectorWritable vector : vectorList) {
       context.write(center, vector);
 }

 if (center.converged(key))
       context.getCounter(Counter.CONVERGED).increment(1);



pressWarnings("deprecation")
rride
ected void cleanup(Context context) throws IOException, InterruptedException {
 super.cleanup(context);
 Configuration conf = context.getConfiguration();
 Path outPath = new Path(conf.get("centroid.path"));
 FileSystem fs = FileSystem.get(conf);
 fs.delete(outPath, true);
 try (SequenceFile.Writer out = SequenceFile.createWriter(fs, context.getConfiguration(), outPath,
             ClusterCenter.class, IntWritable.class)) {
       final IntWritable value = new IntWritable(0);
       for (ClusterCenter center : centers) {
             out.append(center, value);
       }
 }
```

We add this to our list of cluster centers

KMeansReducer

```java
public class KMeansReducer extends Reducer<ClusterCenter, VectorWritable, ClusterCenter, VectorWritable> {

    public static enum Counter {
        CONVERGED
    }

    private final List<ClusterCenter> centers = new ArrayList<>();

    @Override
    protected void reduce(ClusterCenter key, Iterable<VectorWritable> values, Context context) throws IOException,
            InterruptedException {

        List<VectorWritable> vectorList = new ArrayList<>();
        DoubleVector newCenter = null;
        for (VectorWritable value : values) {
            vectorList.add(new VectorWritable(value));
            if (newCenter == null)
                newCenter = value.getVector().deepCopy();
            else
                newCenter = newCenter.add(value.getVector());
        }

        newCenter = newCenter.divide(vectorList.size());
        ClusterCenter center = new ClusterCenter(newCenter);
        centers.add(center);
```

```java
for (VectorWritable vector : vectorList) {
 context.write(center, vector);
}
```

```java
        if (center.converged(key))
            context.getCounter(Counter.CONVERGED).increment(1);

    }

    @SuppressWarnings("deprecation")
    @Override
    protected void cleanup(Context context) throws IOException, InterruptedException {
        super.cleanup(context);
        Configuration conf = context.getConfiguration();
        Path outPath = new Path(conf.get("centroid.path"));
        FileSystem fs = FileSystem.get(conf);
        fs.delete(outPath, true);
        try (SequenceFile.Writer out = SequenceFile.createWriter(fs, context.getConfiguration(), outPath,
                ClusterCenter.class, IntWritable.class)) {
            final IntWritable value = new IntWritable(0);
            for (ClusterCenter center : centers) {
                out.append(center, value);
            }
        }
    }
}
```

We write out the Cluster Center, the data point

KMeansReducer

```
        List<VectorWritable> vectorList = new ArrayList<>();
        DoubleVector newCenter = null;
        for (VectorWritable value : values) {
            vectorList.add(new VectorWritable(value));
            if (newCenter == null)
                newCenter = value.getVector().deepCopy();
            else
                newCenter = newCenter.add(value.getVector());
        }

        newCenter = newCenter.divide(vectorList.size());
        ClusterCenter center = new ClusterCenter(newCenter);
        centers.add(center);
        for (VectorWritable vector : vectorList) {
            context.write(center, vector);


}

        if (center.converged(key))
            context.getCounter(Counter.CONVERGED).increment(1);

    }

    @SuppressWarnings("deprecation")
    @Override
    protected void cleanup(Context context) throws IOException, InterruptedException {
        super.cleanup(context);
        Configuration conf = context.getConfiguration();
        Path outPath = new Path(conf.get("centroid.path"));
        FileSystem fs = FileSystem.get(conf);
        fs.delete(outPath, true);
        try (SequenceFile.Writer out = SequenceFile.createWriter(fs, context.getConfiguration(), outPath,
                ClusterCenter.class, IntWritable.class)) {
            final IntWritable value = new IntWritable(0);
            for (ClusterCenter center : centers) {
                out.append(center, value);
            }
        }
    }
}
```

We check for convergence and increment the counter

Recall that converged() returns a boolean that's true if the 2 centers are different

KMeansReducer

```
            for (VectorWritable value : values) {
                vectorList.add(new VectorWritable(value));
                if (newCenter == null)
                    newCenter = value.getVector().deepCopy();
                else
                    newCenter = newCenter.add(value.getVector());
            }

            newCenter = newCenter.divide(vectorList.size());
            ClusterCenter center = new ClusterCenter(newCenter);
            centers.add(center);
            for (VectorWritable vector : vectorList) {
                context.write(center, vector);
    }

            if (center.converged(key))
                context.getCounter(Counter.CONVERGED).increment(1);

        }

        @SuppressWarnings("deprecation")
        @Override
        protected void cleanup(Context context) throws
    IOException, InterruptedException {
            super.cleanup(context);
            Configuration conf = context.getConfiguration();
            Path outPath = new Path(conf.get("centroid.path"));
            FileSystem fs = FileSystem.get(conf);
            fs.delete(outPath, true);
            try (SequenceFile.Writer out = SequenceFile.createWriter(fs,
    context.getConfiguration(), outPath,
                    ClusterCenter.class, IntWritable.class)) {
                final IntWritable value = new IntWritable(0);
                for (ClusterCenter center : centers) {
                    out.append(center, value);
                }
            }
        }
    }
```

A bit of cleanup at the end

```
        for (VectorWritable value : values) {
            vectorList.add(new VectorWritable(value));
            if (newCenter == null)
                newCenter = value.getVector().deepCopy();
            else
                newCenter = newCenter.add(value.getVector());
        }
```

KMeansReducer

```
        newCenter = newCenter.divide(vectorList.size());
        ClusterCenter center = new ClusterCenter(newCenter);
        centers.add(center);
        for (VectorWritable vector : vectorList) {
            context.write(center, vector);
}

        if (center.converged(key))
            context.getCounter(Counter.CONVERGED).increment(1);

    }
    @SuppressWarnings("deprecation")
    @Override
    protected void cleanup(Context context) throws
IOException, InterruptedException {
        super.cleanup(context);
        Configuration conf = context.getConfiguration();
        Path outPath = new Path(conf.get("centroid.path"));
        FileSystem fs = FileSystem.get(conf);
        fs.delete(outPath, true);
        try (SequenceFile.Writer out = SequenceFile.createWriter(fs,
context.getConfiguration(), outPath,
                ClusterCenter.class, IntWritable.class)) {
            final IntWritable value = new IntWritable(0);
            for (ClusterCenter center : centers) {
                out.append(center, value);
            }
        }
    }
}
```

We write our centers back to file to be read in the next iteration

KMeansReducer

```
        for (VectorWritable value : values) {
            vectorList.add(new VectorWritable(value));
            if (newCenter == null)
                newCenter = value.getVector().deepCopy();
            else
                newCenter = newCenter.add(value.getVector());
        }

        newCenter = newCenter.divide(vectorList.size());
        ClusterCenter center = new ClusterCenter(newCenter);
        centers.add(center);
        for (VectorWritable vector : vectorList) {
            context.write(center, vector);
}

        if (center.converged(key))
            context.getCounter(Counter.CONVERGED).increment(1);

    }

    @SuppressWarnings("deprecation")
    @Override
    protected void cleanup(Context context) throws
IOException, InterruptedException {
        super.cleanup(context);
        Configuration conf = context.getConfiguration();
        Path outPath = new Path(conf.get("centroid.path"));
        FileSystem fs = FileSystem.get(conf);
        fs.delete(outPath, true);
        try (SequenceFile.Writer out = SequenceFile.createWriter(fs,
context.getConfiguration(), outPath,
                ClusterCenter.class, IntWritable.class)) {
            final IntWritable value = new IntWritable(0);
            for (ClusterCenter center : centers) {
                out.append(center, value);
            }
        }
    }
}
```

This was 1 full iteration

KMeansReducer

```
        for (VectorWritable value : values) {
            vectorList.add(new VectorWritable(value));
            if (newCenter == null)
                newCenter = value.getVector().deepCopy();
            else
                newCenter = newCenter.add(value.getVector());
        }

        newCenter = newCenter.divide(vectorList.size());
        ClusterCenter center = new ClusterCenter(newCenter);
        centers.add(center);
        for (VectorWritable vector : vectorList) {
            context.write(center, vector);
}

        if (center.converged(key))
            context.getCounter(Counter.CONVERGED).increment(1);

    }

    @SuppressWarnings("deprecation")
    @Override
    protected void cleanup(Context context) throws
IOException, InterruptedException {
        super.cleanup(context);
        Configuration conf = context.getConfiguration();
        Path outPath = new Path(conf.get("centroid.path"));
        FileSystem fs = FileSystem.get(conf);
        fs.delete(outPath, true);
        try (SequenceFile.Writer out = SequenceFile.createWriter(fs,
context.getConfiguration(), outPath,
                ClusterCenter.class, IntWritable.class)) {
            final IntWritable value = new IntWritable(0);
            for (ClusterCenter center : centers) {
                out.append(center, value);
            }
        }
    }
}
```

The job is complete, let's now go back to the Main class

KMeansClusteringJob

```
        fs.delete(in, true);
    }

    writeExampleCenters(conf, center, fs);
    writeExampleVectors(conf, in, fs);

    FileOutputFormat.setOutputPath(job, out);
    job.setInputFormatClass(SequenceFileInputFormat.class);
    job.setOutputFormatClass(SequenceFileOutputFormat.class);

    job.setOutputKeyClass(ClusterCenter.class);
    job.setOutputValueClass(VectorWritable.class);

    job.waitForCompletion(true);
```

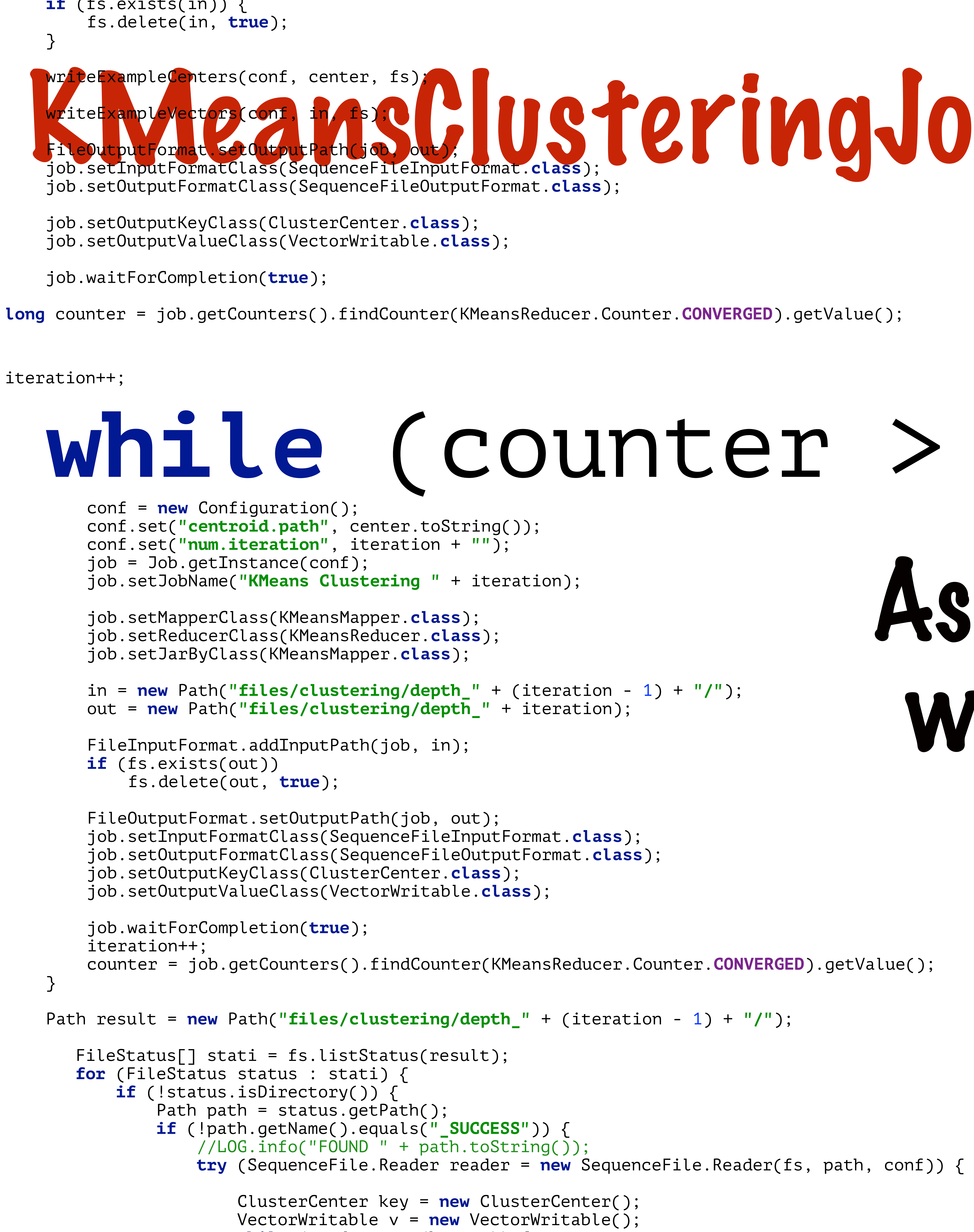
```
long counter =
job.getCounters().findCounter(KMeansReducer.Counter.CONVERGED).getValue();

iteration++;
```

```
    while (counter > 0) {
        conf = new Configuration();
        conf.set("centroid.path", center.toString());
        conf.set("num.iteration", iteration + "");
        job = Job.getInstance(conf);
        job.setJobName("KMeans Clustering " + iteration);

        job.setMapperClass(KMeansMapper.class);
        job.setReducerClass(KMeansReducer.class);
        job.setJarByClass(KMeansMapper.class);

        in = new Path("files/clustering/depth_" + (iteration - 1) + "/");
        out = new Path("files/clustering/depth_" + iteration);

        FileInputFormat.addInputPath(job, in);
        if (fs.exists(out))
            fs.delete(out, true);

        FileOutputFormat.setOutputPath(job, out);
        job.setInputFormatClass(SequenceFileInputFormat.class);
        job.setOutputFormatClass(SequenceFileOutputFormat.class);
        job.setOutputKeyClass(ClusterCenter.class);
        job.setOutputValueClass(VectorWritable.class);

        job.waitForCompletion(true);
        iteration++;
        counter = job.getCounters().findCounter(KMeansReducer.Counter.CONVERGED).getValue();
    }

    Path result = new Path("files/clustering/depth_" + (iteration - 1) + "/");

       FileStatus[] stati = fs.listStatus(result);
       for (FileStatus status : stati) {
           if (!status.isDirectory()) {
               Path path = status.getPath();
               if (!path.getName().equals("_SUCCESS")) {
                   //LOG.info("FOUND " + path.toString());
                   try (SequenceFile.Reader reader = new SequenceFile.Reader(fs, path, conf)) {

                       ClusterCenter key = new ClusterCenter();
```

The job gets back the value of the counter

The iteration number is also updated

KMeansClusteringJob



```
  if (fs.exists(in)) {
    fs.delete(in, true);
  }

  writeExampleCenters(conf, center, fs);

  writeExampleVectors(conf, in, fs);

  FileOutputFormat.setOutputPath(job, out);
  job.setInputFormatClass(SequenceFileInputFormat.class);
  job.setOutputFormatClass(SequenceFileOutputFormat.class);

  job.setOutputKeyClass(ClusterCenter.class);
  job.setOutputValueClass(VectorWritable.class);

  job.waitForCompletion(true);

long counter = job.getCounters().findCounter(KMeansReducer.Counter.CONVERGED).getValue();


iteration++;
  while (counter > 0) {
    conf = new Configuration();
    conf.set("centroid.path", center.toString());
    conf.set("num.iteration", iteration + "");
    job = Job.getInstance(conf);
    job.setJobName("KMeans Clustering " + iteration);

    job.setMapperClass(KMeansMapper.class);
    job.setReducerClass(KMeansReducer.class);
    job.setJarByClass(KMeansMapper.class);

    in = new Path("files/clustering/depth_" + (iteration - 1) + "/");
    out = new Path("files/clustering/depth_" + iteration);

    FileInputFormat.addInputPath(job, in);
    if (fs.exists(out))
      fs.delete(out, true);

    FileOutputFormat.setOutputPath(job, out);
    job.setInputFormatClass(SequenceFileInputFormat.class);
    job.setOutputFormatClass(SequenceFileOutputFormat.class);
    job.setOutputKeyClass(ClusterCenter.class);
    job.setOutputValueClass(VectorWritable.class);

    job.waitForCompletion(true);
    iteration++;
    counter = job.getCounters().findCounter(KMeansReducer.Counter.CONVERGED).getValue();
  }

  Path result = new Path("files/clustering/depth_" + (iteration - 1) + "/");

   FileStatus[] stati = fs.listStatus(result);
   for (FileStatus status : stati) {
     if (!status.isDirectory()) {
       Path path = status.getPath();
       if (!path.getName().equals("_SUCCESS")) {
         //LOG.info("FOUND " + path.toString());
         try (SequenceFile.Reader reader = new SequenceFile.Reader(fs, path, conf)) {

           ClusterCenter key = new ClusterCenter();
           VectorWritable v = new VectorWritable();
```

As long as the counter >0, the whole thing starts up again

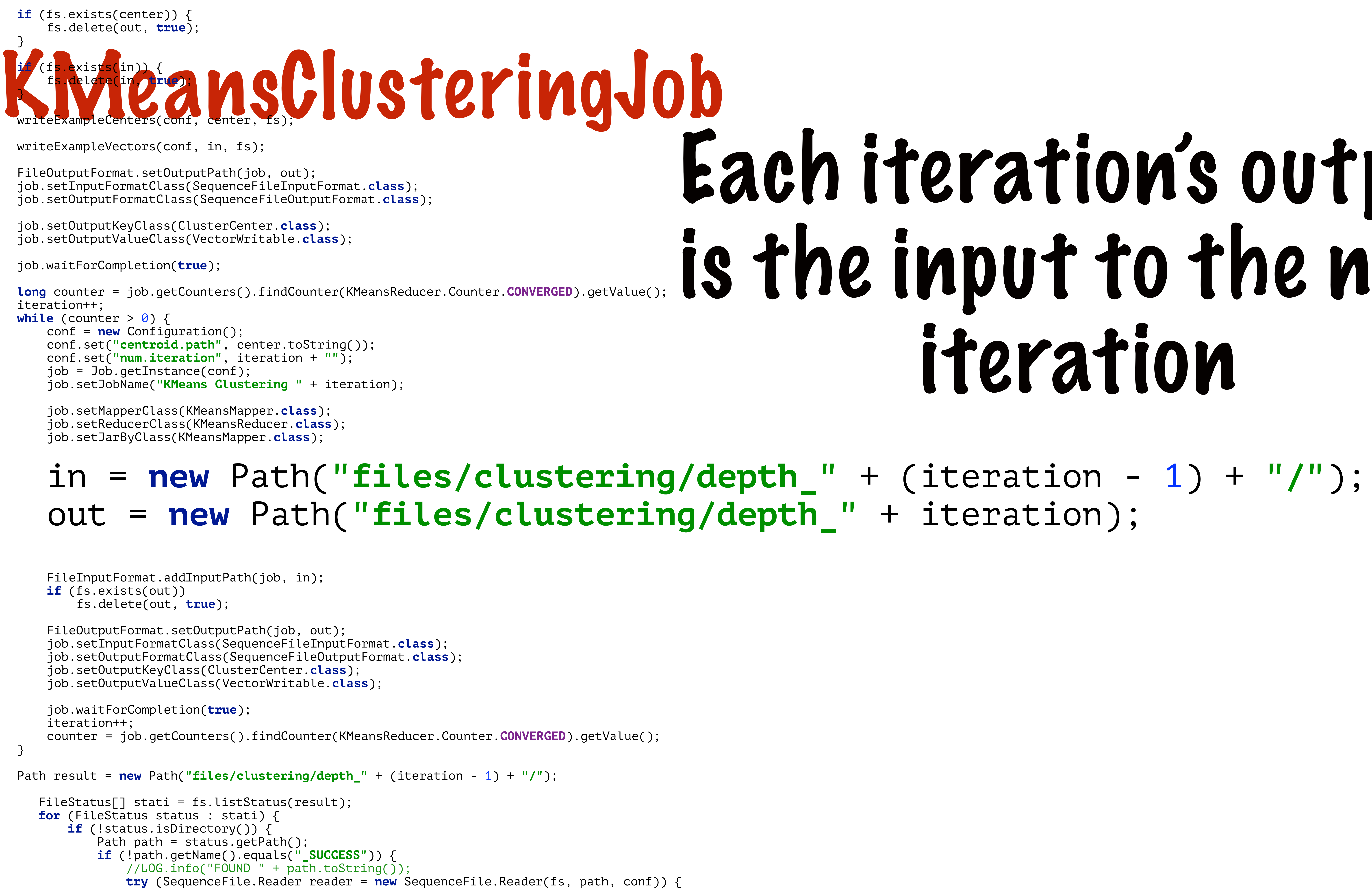
KMeansClusteringJob

```
    FileOutputFormat.setOutputPath(job, out);
    job.setInputFormatClass(SequenceFileInputFormat.class);
    job.setOutputFormatClass(SequenceFileOutputFormat.class);

    job.setOutputKeyClass(ClusterCenter.class);
    job.setOutputValueClass(VectorWritable.class);

    job.waitForCompletion(true);

long counter = job.getCounters().findCounter(KMeansReducer.Counter.CONVERGED).getValue();


iteration++;
  while (counter > 0) {
      conf = new Configuration();
      conf.set("centroid.path", center.toString());
      conf.set("num.iteration", iteration + "");
      job = Job.getInstance(conf);
      job.setJobName("KMeans Clustering " + iteration);

      job.setMapperClass(KMeansMapper.class);
      job.setReducerClass(KMeansReducer.class);
      job.setJarByClass(KMeansMapper.class);

      in = new Path("files/clustering/depth_" + (iteration - 1) + "/");
      out = new Path("files/clustering/depth_" + iteration);

      FileInputFormat.addInputPath(job, in);
      if (fs.exists(out))
          fs.delete(out, true);

      FileOutputFormat.setOutputPath(job, out);
      job.setInputFormatClass(SequenceFileInputFormat.class);
      job.setOutputFormatClass(SequenceFileOutputFormat.class);
      job.setOutputKeyClass(ClusterCenter.class);
      job.setOutputValueClass(VectorWritable.class);

      job.waitForCompletion(true);

      iteration++;
      counter =
job.getCounters().findCounter(KMeansReducer.Counter.CONVERGED).getValue();
    }

    Path result = new Path("files/clustering/depth_" + (iteration - 1) + "/");

      FileStatus[] stati = fs.listStatus(result);
      for (FileStatus status : stati) {
          if (!status.isDirectory()) {
              Path path = status.getPath();
              if (!path.getName().equals("_SUCCESS")) {
                  //LOG.info("FOUND " + path.toString());
                  try (SequenceFile.Reader reader = new SequenceFile.Reader(fs, path, conf)) {
```

Another job is setup and started

KMeansClusteringJob

Each iteration's output is the input to the new iteration



```
if (fs.exists(center)) {
    fs.delete(out, true);
}

if (fs.exists(in)) {
    fs.delete(in, true);
}

writeExampleCenters(conf, center, fs);

writeExampleVectors(conf, in, fs);

FileOutputFormat.setOutputPath(job, out);
job.setInputFormatClass(SequenceFileInputFormat.class);
job.setOutputFormatClass(SequenceFileOutputFormat.class);

job.setOutputKeyClass(ClusterCenter.class);
job.setOutputValueClass(VectorWritable.class);

job.waitForCompletion(true);

long counter = job.getCounters().findCounter(KMeansReducer.Counter.CONVERGED).getValue();
iteration++;
while (counter > 0) {
    conf = new Configuration();
    conf.set("centroid.path", center.toString());
    conf.set("num.iteration", iteration + "");
    job = Job.getInstance(conf);
    job.setJobName("KMeans Clustering " + iteration);

    job.setMapperClass(KMeansMapper.class);
    job.setReducerClass(KMeansReducer.class);
    job.setJarByClass(KMeansMapper.class);

    in = new Path("files/clustering/depth_" + (iteration - 1) + "/");
    out = new Path("files/clustering/depth_" + iteration);


    FileInputFormat.addInputPath(job, in);
    if (fs.exists(out))
        fs.delete(out, true);

    FileOutputFormat.setOutputPath(job, out);
    job.setInputFormatClass(SequenceFileInputFormat.class);
    job.setOutputFormatClass(SequenceFileOutputFormat.class);
    job.setOutputKeyClass(ClusterCenter.class);
    job.setOutputValueClass(VectorWritable.class);

    job.waitForCompletion(true);
    iteration++;
    counter = job.getCounters().findCounter(KMeansReducer.Counter.CONVERGED).getValue();
}

Path result = new Path("files/clustering/depth_" + (iteration - 1) + "/");

   FileStatus[] stati = fs.listStatus(result);
   for (FileStatus status : stati) {
       if (!status.isDirectory()) {
           Path path = status.getPath();
           if (!path.getName().equals("_SUCCESS")) {
               //LOG.info("FOUND " + path.toString());
               try (SequenceFile.Reader reader = new SequenceFile.Reader(fs, path, conf)) {
```

KMeansClusteringJob

```
    job.setOutputFormatClass(SequenceFileOutputFormat.class);

    job.setOutputKeyClass(ClusterCenter.class);
    job.setOutputValueClass(VectorWritable.class);

    job.waitForCompletion(true);

long counter = job.getCounters().findCounter(KMeansReducer.Counter.CONVERGED).getValue();

iteration++;
    while (counter > 0) {
        conf = new Configuration();
        conf.set("centroid.path", center.toString());
        conf.set("num.iteration", iteration + "");
        job = Job.getInstance(conf);
        job.setJobName("KMeans Clustering " + iteration);

        job.setMapperClass(KMeansMapper.class);
        job.setReducerClass(KMeansReducer.class);
        job.setJarByClass(KMeansMapper.class);

        in = new Path("files/clustering/depth_" + (iteration - 1) + "/");
        out = new Path("files/clustering/depth_" + iteration);

        FileInputFormat.addInputPath(job, in);
        if (fs.exists(out))
            fs.delete(out, true);

        FileOutputFormat.setOutputPath(job, out);
        job.setInputFormatClass(SequenceFileInputFormat.class);
        job.setOutputFormatClass(SequenceFileOutputFormat.class);
        job.setOutputKeyClass(ClusterCenter.class);
        job.setOutputValueClass(VectorWritable.class);

        job.waitForCompletion(true);
        iteration++;
        counter =
job.getCounters().findCounter(KMeansReducer.Counter.CONVERGED).getValue();
    }

    Path result = new Path("files/clustering/depth_" + (iteration - 1) + "/");

       FileStatus[] stati = fs.listStatus(result);
       for (FileStatus status : stati) {
           if (!status.isDirectory()) {
               Path path = status.getPath();
               if (!path.getName().equals("_SUCCESS")) {
                   //LOG.info("FOUND " + path.toString());
                   try (SequenceFile.Reader reader = new SequenceFile.Reader(fs, path, conf)) {

                       ClusterCenter key = new ClusterCenter();
                       VectorWritable v = new VectorWritable();
                       while (reader.next(key, v)) {
                          // LOG.info(key + " / " + v);
                           System.out.println( key + " /" + v);
```

When the centers don't change anymore, the loop breaks out

KMeansClusteringJob

```
        job.setOutputFormatClass(SequenceFileOutputFormat.class);
        job.setOutputKeyClass(ClusterCenter.class);
        job.setOutputValueClass(VectorWritable.class);

        job.waitForCompletion(true);
        iteration++;
        counter = job.getCounters().findCounter(KMeansReducer.Counter.CONVERGED).getValue();
  }

  Path result = new Path("files/clustering/depth_" + (iteration - 1) + "/");

        FileStatus[] stati = fs.listStatus(result);
        for (FileStatus status : stati) {
            if (!status.isDirectory()) {
                Path path = status.getPath();
                if (!path.getName().equals("_SUCCESS")) { try (SequenceFile.Read
reader = new SequenceFile.Reader(fs, path, conf)) {

                        ClusterCenter key = new ClusterCenter();
                        VectorWritable v = new VectorWritable();
                        while (reader.next(key, v)) {
                          System.out.println( key + " /" + v);

                        }
                    }
                }
            }
        }
}
```

We read the last iteration's output and print it out to screen