When reading input, or writing output from a MapReduce application, it is sometimes easier to work with data using an abstract class instead of the primitive Hadoop Writable classes (for example, `Text` and `IntWritable`). This recipe demonstrates how to create a custom Hadoop Writable and InputFormat that can be used by MapReduce applications.

How it works...

The first task was to define our own Hadoop key and value representations by implementing the `WritableComparable` interface. The `WritableComparable` interface allows us to create our own abstract types, which can be used as keys or values by the MapReduce framework.

Next, we created an InputFormat that inherits from the `FileInputFormat` class. The Hadoop `FileInputFormat` is the base class for all file-based InputFormats. The InputFormat takes care of managing the input files for a MapReduce job. Since we do not want to change the way in which our input files are split and distributed across the cluster, we only need to override two methods, `createRecordReader()` and `isSplitable()`.

The `isSplitable()` method is used to instruct the `FileInputFormat` class that it is acceptable to split up the input files if there is a codec available in the Hadoop environment to read and split the file. The `createRecordReader()` method is used to create a Hadoop RecordReader that processes individual file splits and generates a key-value pair for the mappers to process.

After the `GeoInputFormat` class was written, we wrote a RecordReader to process the individual input splits and create `GeoKey` and `GeoValue` for the mappers. The `GeoRecordReader` class reused the Hadoop `LineRecordReader` class to read from the input split. When the `LineRecordReader` class completed reading a record from the `Nigeria_ACLED_cleaned.tsv` dataset, we created two objects. These objects are `GeoKey` and `GeoValue`, which are sent to the mapper.