```
import java.io.IOException;

import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.util.Bytes;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.mapred.lib.NullOutputFormat;
import org.apache.hadoop.util.*;

/**
 * A MapReduce application to count the number of rows in an HBase table
 */
public class RowCounter extends Configured implements Tool {
// Name of this program
static final String NAME = "rowcounter";

        /* HBase TableMap interface is a specialization of org.apache.hadoop.mapred.Mapper
         * that sets the map input types passed by TableInputFormat (set by JobConf below via
TableMapUtil.initTableMapJob() utility method
         * this mapper checks all the columns of a row, if all are empty, it doesn't count the row,
otherwise it increments Counters.ROWS by one
         */
        static class RowCounterMapper implements TableMap<ImmutableBytesWritable,
RowResult> {
                private static enum Counters {ROWS}

                public void map(ImmutableBytesWritable row, RowResult value,
OutputCollector<ImmutableBytesWritable, RowResult> output,
                        Reporter reporter) throws IOException {
                        boolean content = false;
                        for (Map.Entry<byte [], Cell> e: value.entrySet(){
                        Cell cell = e.getValue();
                                if (cell != null && cell.getValue().length > 0) {
                                        content = true;
                                        break;
                                }
                        }
                        if (!content) {
                                // Don't count rows that are all empty values.
                                return;
                        }
                        // Give out some value every time. We are only interested in the row/key
                        reporter.incrCounter(Counters.ROWS, 1);
                }

                public void configure (JobConf jc) {
                        // Nothing to do.
```

```
		}

		public void close() throws IOException {
			// Nothing to do.
		}
	}

	/*
	 *This method parses arguments added to the configuration that were passed on the
command line figuring the table and columns
	 * to run RowCounter against. It also invokes TableMapUtil.initTableMapJob() utility
method,
	 * which among other things such as setting the map class to use, sets the input format
to TableInputFormat.
	 */
	public JobConf createSubmittableJob(String[] args) throws IOException {
		JobConf c = new JobConf(getConf(), getClass());
		c.setJobName(NAME);
		// Columns are space delimited
		StringBuilder sb = new StringBuilder();
		final int column offset = 2;
		for (int i = columnoffset; i < args.length, i++) {
			if (i > columnoffset) {
				sb.append(" ");
			}
			sb.append(args[i]);
		}
		// Second argument is the table name.
		TableMapReduceUtil.initTableMapJob(args[1], sb.toString(),
RowCounterMapper.class, ImmutableBytesWritable.class,
			RowResult.class, c);
		c.setNumReduceTasks(0);
		// First arg is the output directory.
		FileOutputFormat.setOutputPath(c, new Path(args[0]));
		return c;
	}

	static int printUsage() {
		System.out.prinln(NAME + " <outputdir> <tablename> <column1>
[<column2>...]");
		return -1;
	}

	/**
	 * run() method belongs to Tool implementation
	 */
	public int run(final String[] args) throws Exception {
		// Make sure there are at least 3 parameters
		if (args.length < 3) {
			System.err.println("ERROR: Wrong number of parameters: " +
args.length);
```

```
                        return printUsage();
                }
                JobClient.runJob(createSubmitableJob(args));
                return 0;
        }

        /**
         * RowCounter's main() method does not invoke its own run() method directly. Instead it
calls ToolRunner's static run() method,
         * which takes care of creating a Configuration object for the Tool, before calling its run()
method. ToolRunner also uses a
         * GenericOptionsParser to pick up any standard options specified on the command line,
and set them on the Configuration instance.
         */
        public static void main(String[] args) throws Exception {
                HBaseConfiguration c = new HBaseConfiguration();
                int errCode = ToolRunner.run(c, new RowCounter(), args);
                System.exit(errCode);
        }
}
```