```
package fm.last.stats;

import java.io.IOException;
import java.util.HashSet;
import java.util.Set;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;



public class LastFMConstants {

    public static final int USER_ID = 0;
    public static final int TRACK_ID = 1;
    public static final int IS_SHARED = 2;
    public static final int RADIO = 3;
    public static final int IS_SKIPPED = 4;

}

public static class UniqueListenersMapper extends
Mapper< Object , Text, IntWritable, IntWritable > {
        IntWritable trackId = new IntWritable();
        IntWritable userId = new IntWritable();

public void map(Object key, Text value,
    Mapper< Object, Text, IntWritable, IntWritable > .Context context)
        throws IOException, InterruptedException {

    String[] parts = value.toString().split("[|]");
    trackId.set(Integer.parseInt(parts[LastFMConstants.TRACK_ID]));
    userId.set(Integer.parseInt(parts[LastFMConstants.USER_ID]));
        if (parts.length == 5) {
        context.write(trackId, userId);
    } else {
        // add counter for invalid records
        context.getCounter(COUNTERS.INVALID_RECORD_COUNT).increment(1L);
    }
    }
}

public static class UniqueListenersReducer extends
    Reducer< IntWritable , IntWritable, IntWritable, IntWritable> {
    public void reduce(
        IntWritable trackId,
        Iterable< IntWritable > userIds,
        Reducer< IntWritable , IntWritable, IntWritable,
        IntWritable>.Context context)
```

```
        throws IOException, InterruptedException {
        Set< Integer > userIdSet = new HashSet< Integer >();
        for (IntWritable userId : userIds) {
        userIdSet.add(userId.get());
        }
        IntWritable size = new IntWritable(userIdSet.size());
        context.write(trackId, size);
    }
}
public static void main(String[] args) throws Exception {
        Configuration <span id="IL_AD9" class="IL_AD">conf</span> = new Configuration()
        if (args.length != 2) {
            System.err.println("Usage: uniquelisteners < in > < out >");
            System.exit(2);
        }
        Job job = new Job(conf, "Unique listeners per track");
        job.setJarByClass(UniqueListeners.class);
        job.setMapperClass(UniqueListenersMapper.class);
        job.setReducerClass(UniqueListenersReducer.class);
        job.setOutputKeyClass(IntWritable.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
        org.apache.hadoop.mapreduce.Counters counters = job.getCounters();
        System.out.println("No. of Invalid Records :"
                + counters.findCounter(COUNTERS.INVALID_RECORD_COUNT)
                        .getValue());
    }
```

```
package fm.lastskipped.stats;

import java.io.IOException;
import java.util.HashSet;
import java.util.Set;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;



public class LastFMConstants {

    public static final int USER_ID = 0;
    public static final int TRACK_ID = 1;
    public static final int IS_SHARED = 2;
    public static final int RADIO = 3;
    public static final int IS_SKIPPED = 4;

}

public static class FullyHeardMapper extends
Mapper< Object , Text, IntWritable, IntWritable > {
        IntWritable trackId = new IntWritable();
        IntWritable IS_SKIPPED = new IntWritable();

public void map(Object key, Text value,
    Mapper< Object, Text, IntWritable, IntWritable > .Context context)
        throws IOException, InterruptedException {

    String[] parts = value.toString().split("[|]");
    trackId.set(Integer.parseInt(parts[LastFMConstants.TRACK_ID]));
    IS_SKIPPED.set(Integer.parseInt(parts[LastFMConstants.IS_SKIPPED]));
        if (parts.length == 5) {
        context.write(trackId, IS_SKIPPED);
    } else {
        // add counter for invalid records
        context.getCounter(COUNTERS.INVALID_RECORD_COUNT).increment(1L);
    }
    }
}

public static class FullyHeardReducer extends
    Reducer< IntWritable , IntWritable, IntWritable, IntWritable> {
    public void reduce(
        IntWritable trackId,
        Iterable< IntWritable > IS_SKIPPED,
        Reducer< IntWritable , IntWritable, IntWritable,
        IntWritable>.Context context)
```

```
        throws IOException, InterruptedException {
        Set< Integer > trackId = new HashSet< Integer >();
        for (IntWritable IS_SKIPPED=1) {
        userIdSet.add(userId.get());
        }
        IntWritable size = new IntWritable(userIdSet.size());
        context.write(IS_SKIPPED, size);
    }
}
public static void main(String[] args) throws Exception {
  Configuration <span id="IL_AD9" class="IL_AD">conf</span> = new Configuration();
        if (args.length != 2) {
            System.err.println("Usage: Skipped Songs < in > < out >");
            System.exit(2);
        }
        Job job = new Job(conf, "Fully heard per track");
        job.setJarByClass(FullyHeard.class);
        job.setMapperClass(FullyHeardMapper.class);
        job.setReducerClass(FullyHeardReducer.class);
        job.setOutputKeyClass(IntWritable.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
        org.apache.hadoop.mapreduce.Counters counters = job.getCounters();
        System.out.println("No. of Invalid Records :"
                + counters.findCounter(COUNTERS.INVALID_RECORD_COUNT)
                        .getValue());
    }
```

```
package fm.shared.stats;

import java.io.IOException;
import java.util.HashSet;
import java.util.Set;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;



public class SharedFMConstants {

    public static final int USER_ID = 0;
    public static final int TRACK_ID = 1;
    public static final int IS_SHARED = 2;
    public static final int RADIO = 3;
    public static final int IS_SKIPPED = 4;

}

public static class SharedFMMapper extends
Mapper< Object , Text, IntWritable, IntWritable > {
        IntWritable trackId = new IntWritable();
        IntWritable IS_SHARED = new IntWritable();

public void map(Object key, Text value,
    Mapper< Object, Text, IntWritable, IntWritable > .Context context)
        throws IOException, InterruptedException {

    String[] parts = value.toString().split("[|]");
    trackId.set(Integer.parseInt(parts[LastFMConstants.TRACK_ID]));
    IS_SHARED.set(Integer.parseInt(parts[LastFMConstants.IS_SHARED]));
        if (parts.length == 5) {
        context.write(trackId, IS_SHARED);
    } else {
        // add counter for invalid records
        context.getCounter(COUNTERS.INVALID_RECORD_COUNT).increment(1L);
    }
    }
}

public static class SharedFMReducer extends
    Reducer< IntWritable , IntWritable, IntWritable, IntWritable> {
    public void reduce(
        IntWritable trackId,
        Iterable< IntWritable > IS_SHARED,
        Reducer< IntWritable , IntWritable, IntWritable,
```

```
        IntWritable>.Context context)
        throws IOException, InterruptedException {
        Set< Integer > trackId = new HashSet< Integer >();
        for (IntWritable IS_SHARED=1) {
        userIdSet.add(userId.get());
        }
        IntWritable size = new IntWritable(userIdSet.size());
        context.write(IS_SHARED, size);
    }
}
public static void main(String[] args) throws Exception {
  Configuration <span id="IL_AD9" class="IL_AD">conf</span> = new Configuration();
        if (args.length != 2) {
            System.err.println("Usage: SHARED < in > < out >");
            System.exit(2);
        }
        Job job = new Job(conf, "Shared track");
        job.setJarByClass(SharedFM.class);
        job.setMapperClass(SharedFMMapper.class);
        job.setReducerClass(SharedFMReducer.class);
        job.setOutputKeyClass(IntWritable.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
        org.apache.hadoop.mapreduce.Counters counters = job.getCounters();
        System.out.println("No. of Invalid Records :"
                + counters.findCounter(COUNTERS.INVALID_RECORD_COUNT)
                        .getValue());
    }
```