Name: MIHIR SONI BHARATHAI

ROLL NO: 202300819010048

Q-1: Mapper class

```
import java.io.IOExcep*on; import
org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
importorg.apache.hadoop.mapreduce.Mapper;

public class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable>{

 private final static IntWritable one = new IntWritable(1);
private Text word = new Text();

   // The map method
   public void map(Object key, Text value, Context context) throws IOExcep*on,
InterruptedExcep*on
{
      // Split the line into tokens (words)
      String[] tokens = value.toString().split("\\s+");

      // Iterate through each word in the tokens array
 for (String token : tokens)
{          word.set(token);  context.write(word, one);  // Emit the word as the
key and 1 as the value
      }
   }
}
```

Reducer class:

```
import java.io.IOExcep*on;
```

```
import org.apache.hadoop.io.IntWritable; import
org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

   private IntWritable result = new IntWritable();

   // The reduce method
   public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
IOExcep*on, InterruptedExcep*on {
      int sum = 0;
      // Iterate over the values (which are all 1s) and sum them up       for
(IntWritable val : values) {          sum += val.get();
      }
      result.set(sum);
      // Emit the word and its count       context.write(key,
result);
   }
}
```

Driver class:

```
import org.apache.hadoop.conf.Configura*on;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
importorg.apache.hadoop.mapreduce.lib.input.FileInputFormat;
importorg.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.u*l.GenericOp*onsParser;

public class WordCount {
```

```
  public staticc void main(String[] args) throws Excep*on {
    Configura*on conf = new Configura*on();
    String[] otherArgs = new GenericOp*onsParser(conf, args).getRemainingArgs();

    if (otherArgs.length < 2) {
      System.err.println("Usage: wordcount <in> <out>");
      System.exit(2);
    }

 Job job = new Job(conf, "word count");
job.setJarByClass(WordCount.class);
job.setMapperClass(TokenizerMapper.class);
job.setCombinerClass(IntSumReducer.class);
job.setReducerClass(IntSumReducer.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);

    // Set input and output paths
    FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));

    System.exit(job.waitForComple*on(true) ? 0 : 1);
  }
}
```

Q-2: Mapper class:

```
import java.io.IOExcep*on;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
```

```
public class MinTempMapper extends Mapper<Object, Text, Text, IntWritable> {


   private Text year = new Text();     private IntWritable
temperature = new IntWritable();


   public void map(Object key, Text value, Context context) throws IOExcep*on,
InterruptedExcep*on
{
      // Split the line into year and temperature
      String[] fields = value.toString().split("\\s+");


      // Parse the year and temperature        if (fields.length == 2) {
year.set(fields[0]);  // Set the year
temperature.set(Integer.parseInt(fields[1]));  // Set the temperature
         // Emit the year as the key and temperature as the value
context.write(year, temperature);
      }
   }
}
```

Reducer class:

```
import java.io.IOExcep*on;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
public class MinTempReducer extends
Reducer<Text, IntWritable, Text, IntWritable> {


   private IntWritable result = new IntWritable();
```

```
   public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
IOExcep*on, InterruptedExcep*on {        int minTemp = Integer.MAX_VALUE;

      // Iterate through all temperatures and find the
minimum        for (IntWritable val : values) {            minTemp
= Math.min(minTemp, val.get());
      }

      result.set(minTemp);
      // Emit the year and the minimum temperature
context.write(key, result);
   }
}
```

Driver class:

```
import org.apache.hadoop.conf.Configura*on; 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.lib.input.FileInputFormat; import
org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.u*l.GenericOp*onsParser;

public class MinTemp {
   public static void main(String[] args) throws Excep*on {
      Configura*on conf = new Configura*on();
      String[] otherArgs = new GenericOp*onsParser(conf, args).getRemainingArgs();

      if (otherArgs.length < 2) {
```

```
        System.err.println("Usage: mintemp <in> <out>");
        System.exit(2);
      }

Job job = new Job(conf, "minimum temperature");
job.setJarByClass(MinTemp.class);
job.setMapperClass(MinTempMapper.class);
job.setReducerClass(MinTempReducer.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);

      // Set input and output paths
      FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
      FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));

      System.exit(job.waitForComple*on(true) ? 0 : 1);
   }
}
```

Text file:

2014 -1

2014 5

2015 20

2015 25

2016 30

2016 28

Q-3: Mapper class:

```
import java.io.IOExcep*on; import
org.apache.hadoop.io.IntWritable; import
org.apache.hadoop.io.Text; import
org.apache.hadoop.mapreduce.Mapper;
```

```
public class TokenCountMapper extends Mapper<Object, Text, Text, IntWritable> {

   private final sta*c IntWritable one = new
IntWritable(1);     private Text word = new Text();
private Text specialKey = new Text("SpecialCount");

   public void map(Object key, Text value, Context context) throws IOExcep*on,
InterruptedExcep*on
{
      // Split the line into tokens (words)
      String[] tokens = value.toString().split("\\s+");
int tokenCount = tokens.length;

      // Emit each word as key with 1 as the
value        for (String token : tokens) {
word.set(token);           context.write(word,
one);
      }

      // Emit a special key with token count for later aggrega*on
context.write(specialKey, new IntWritable(tokenCount));
   }
}
```

Reducer class:

```
import java.io.IOExcep*on;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text; import
org.apache.hadoop.mapreduce.Reducer;

public class TokenCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
```

```
   private IntWritable result = new
IntWritable();     private int totalTokens = 0;
private int lineCount = 0;

   public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOExcep*on,
InterruptedExcep*on {
      int sum = 0;

      // Special case: if the key is "SpecialCount", aggregate token count and line
count        if (key.toString().equals("SpecialCount")) {            for (IntWritable val :
values) {               totalTokens += val.get();               lineCount++;
         }
      } else {
         // For regular tokens, sum up the values to count occurrences of the
token            for (IntWritable val : values) {               sum += val.get();
         }
         result.set(sum);            context.write(key, result);  //
Emit the token and its count
      }
   }

   // ASer all the data is processed, emit the average token count
   @Override     protected void cleanup(Context context) throws IOExcep*on,
InterruptedExcep*on {
      if (lineCount > 0) {            float average = (float) totalTokens / lineCount;
context.write(new Text("AverageCount"), new IntWritable(Math.round(average)));
      }
   }
}
```

Driver class:

```
import org.apache.hadoop.conf.Configura*on; 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.lib.input.FileInputFormat; import
org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.u*l.GenericOp*onsParser;

public class TokenCount {

   public sta*c void main(String[] args) throws Excep*on {
      Configura*on conf = new Configura*on();
      String[] otherArgs = new GenericOp*onsParser(conf, args).getRemainingArgs();

      if (otherArgs.length < 2) {
         System.err.println("Usage: tokencount <in> <out>");
         System.exit(2);
      }

      Job job = new Job(conf, "token count with average");        job.setJarByClass(TokenCount.class);
job.setMapperClass(TokenCountMapper.class);
job.setReducerClass(TokenCountReducer.class);        job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);

      // Set input and output paths
      FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
      FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
```

```
        System.exit(job.waitForComple*on(true) ? 0 : 1);
    }
}
```

Text file:

Hadoop is a framework

Hotspot JVM for Java

Hadoop is great

Q-4: Mapper class:

```
import java.io.IOExcep*on; import
org.apache.hadoop.io.IntWritable; import
org.apache.hadoop.io.Text; import
org.apache.hadoop.mapreduce.Mapper;

public class TokenLengthMapper extends Mapper<Object, Text, Text, IntWritable> {

    private final sta*c IntWritable one = new IntWritable(1);
private Text specialKey = new Text("TokenCount");

    public void map(Object key, Text value, Context context) throws IOExcep*on,
InterruptedExcep*on
{
        // Split the line into tokens (words)
        String[] tokens = value.toString().split("\\s+");

        // Iterate over the tokens
for (String token : tokens) {
if (token.length() >= 4) {
                // Emit "TokenCount" as key and 1 as the value for tokens with length >= 4
context.write(specialKey, one);
```

```
        }
      }
   }
}
```

Reducer class:

```
import java.io.IOExcep*on; import
org.apache.hadoop.io.IntWritable; import
org.apache.hadoop.io.Text; import
org.apache.hadoop.mapreduce.Reducer;

public class TokenLengthReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

   private IntWritable result = new IntWritable();

   public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOExcep*on,
InterruptedExcep*on {
      int sum = 0;

      // Sum up all the values (counts) for tokens with length >= 4
for (IntWritable val : values) {
         sum += val.get();
      }

      result.set(sum);
      // Emit the special key and the total count of tokens with length >= 4
context.write(key, result);
   }
}
```

Driver class:

```
import org.apache.hadoop.conf.Configura*on; 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.lib.input.FileInputFormat; import
org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.u*l.GenericOp*onsParser;

public class TokenLength {

   public sta*c void main(String[] args) throws Excep*on {
      Configura*on conf = new Configura*on();
      String[] otherArgs = new GenericOp*onsParser(conf, args).getRemainingArgs();

      if (otherArgs.length < 2) {
         System.err.println("Usage: tokenlength <in> <out>");
         System.exit(2);
      }

Job job = new Job(conf, "token length count");
 job.setJarByClass(TokenLength.class);
job.setMapperClass(TokenLengthMapper.class);
job.setReducerClass(TokenLengthReducer.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);

      // Set input and output paths
      FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
      FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
```

```
        System.exit(job.waitForComple*on(true) ? 0 : 1);
    }
}
```

Text file:

```
Hadoop is a powerful framework
MapReduce is a processing model
Distributed compu*ng is great
```

Q-5: Mapper class:

```
import java.io.IOExcep*on; import
org.apache.hadoop.io.IntWritable; import
org.apache.hadoop.io.Text; import
org.apache.hadoop.mapreduce.Mapper;

public class FemaleVoterMapper extends Mapper<Object, Text, Text, IntWritable> {

    private final sta*c IntWritable one = new IntWritable(1);
private Text femaleKey = new Text("FemaleVoterCount");

    public void map(Object key, Text value, Context context) throws IOExcep*on,
InterruptedExcep*on
{
        // Skip the header if it's the first line
String line = value.toString();        if
(line.startsWith("ID,NAME,GENDER,AGE")) {
            return;
        }
```

```
      // Split the line by commas to extract fields
String[] fields = line.split(",");       if (fields.length
== 4) {
          String gender = fields[2].trim();  // Get the gender field


          // Check if gender is female (F)
if (gender.equalsIgnoreCase("F")) {
             // Emit the special key for female voters with a value of 1
context.write(femaleKey, one);
          }
       }
   }
}
```

Reducer class:

```
import java.io.IOExcep*on; import
org.apache.hadoop.io.IntWritable; import
org.apache.hadoop.io.Text; import
org.apache.hadoop.mapreduce.Reducer;


public class FemaleVoterReducer extends Reducer<Text, IntWritable, Text, IntWritable> {


   private IntWritable result = new IntWritable();


   public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOExcep*on,
InterruptedExcep*on {
      int sum = 0;


      // Sum all the values to get the total number of female
voters        for (IntWritable val : values) {            sum += val.get();
      }
```

```
        result.set(sum);
        // Emit the final count of female voters
context.write(new Text("No. of female voters are: "), result);
    }
}
```

Driver class:

```
import org.apache.hadoop.conf.Configura*on; 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.lib.input.FileInputFormat; import
org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.u*l.GenericOp*onsParser;

public class FemaleVoterCount {

    public sta*c void main(String[] args) throws Excep*on {
        Configura*on conf = new Configura*on();
        String[] otherArgs = new GenericOp*onsParser(conf, args).getRemainingArgs();

        if (otherArgs.length < 2) {
            System.err.println("Usage: femalevotercount <in> <out>");
            System.exit(2);
        }

        Job job = new Job(conf, "female voter count");
job.setJarByClass(FemaleVoterCount.class);
job.setMapperClass(FemaleVoterMapper.class);
job.setReducerClass(FemaleVoterReducer.class);
```

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

      // Set input and output paths
      FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
      FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));

      System.exit(job.waitForComple*on(true) ? 0 : 1);
   }
}
```

Q-6: Mapper class:

```
import java.io.IOExcep*on; import
org.apache.hadoop.io.IntWritable; import
org.apache.hadoop.io.Text; import
org.apache.hadoop.mapreduce.Mapper;

public class ReviewCountMapper extends Mapper<Object, Text, Text, IntWritable> {

   private final sta*c IntWritable one = new IntWritable(1);
private Text userId = new Text();

   public void map(Object key, Text value, Context context) throws IOExcep*on,
InterruptedExcep*on
{
      // Split the input line by commas to extract fields
      String[] fields = value.toString().split(",");

      // Ensure the line has enough fields
if (fields.length > 0) {
         String reviewerID = fields[0].trim();  // Extract the reviewerID (UserID)
```

```
        // Emit the UserID with a count of 1
userId.set(reviewerID);
context.write(userId, one);
      }
   }
}
```

Reducer class:

```
Import java.io.IOExcep*on;
 Import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
Importorg.apache.hadoop.mapreduce.Reducer;

public class ReviewCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

   private IntWritable result = new IntWritable();

   public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOExcep*on,
InterruptedExcep*on {
      int sum = 0;

      // Sum the number of reviews for each user
      for (IntWritable val : values) {
sum += val.get();
      }

      result.set(sum);
      // Emit the UserID and the total number of reviews
context.write(key, result);
```

```
  }
```

Driver class:

```
import org.apache.hadoop.conf.Configura*on; 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.lib.input.FileInputFormat; import
org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.u*l.GenericOp*onsParser;

public class ReviewCount {

   public sta*c void main(String[] args) throws Excep*on {
      Configura*on conf = new Configura*on();
      String[] otherArgs = new GenericOp*onsParser(conf, args).getRemainingArgs();

      if (otherArgs.length < 2) {
         System.err.println("Usage: reviewcount <in> <out>");
         System.exit(2);
      }

      Job job = new Job(conf, "review count");
 job.setJarByClass(ReviewCount.class);
job.setMapperClass(ReviewCountMapper.class);
job.setReducerClass(ReviewCountReducer.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
```

```
        // Set input and output paths
        FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
        FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));

        System.exit(job.waitForComple*on(true) ? 0 : 1);
    }
}
```

Q-7: Mapper class:

7.1 Write a MapReduce job to display all the details of the comedy movies. import java.io.IOExcep*on; import org.apache.hadoop.io.Text; import org.apache.hadoop.mapreduce.Mapper;

```
public class ComedyMoviesMapper extends Mapper<Object, Text, Text, Text> {

    public void map(Object key, Text value, Context context) throws IOExcep*on,
InterruptedExcep*on
{
        // Skip the header line
String line = value.toString();        i
f (line.startsWith("movieId"))
 return;

        // Split the line into fields
        String[] fields = line.split(",");

        if (fields.length == 3) {
            String genres = fields[2].trim();

            // Check if genres contain "Comedy"
 if (genres.contains("Comedy"))
{
```

```
context.write(new Text(fields[0]), new Text(line));  // movieId as key, full record as value
          }
       }
    }
}
```

Reducer class:

```
import java.io.IOExcep*on; import
org.apache.hadoop.io.Text; import
org.apache.hadoop.mapreduce.Reducer;

public class ComedyMoviesReducer extends Reducer<Text, Text, Text, Text> {

    public void reduce(Text key, Iterable<Text> values, Context context) throws IOExcep*on,
InterruptedExcep*on {
       for (Text value : values)
{
  context.write(key, value);  // Emit the movie details
       }
    }
}
```

7.2 Write a MapReduce job to find the count of the Documentary movies released in the year 1995.

Mapper Class:

```
import java.io.IOExcep*on; import
org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class Documentary1995Mapper extends Mapper<Object, Text, Text, IntWritable> {
```

```
 private final sta*c IntWritable one = new IntWritable(1);
private Text documentaryKey = new Text("Documentary_1995");


   public void map(Object key, Text value, Context context) throws IOExcep*on,
InterruptedExcep*on
{
      // Skip the header line        String
line = value.toString();        if
(line.startsWith("movieId")) return;


      // Split the line into fields
      String[] fields = line.split(",");


      if (fields.length == 3) {
         String *tle = fields[1].trim();
         String genres = fields[2].trim();


         // Check if the genre is Documentary and year is 1995            if
(genres.contains("Documentary") && *tle.contains("(1995)")) {
context.write(documentaryKey, one);
         }
      }
   }
}
```

Reducer class:

```
import java.io.IOExcep*on; import
org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
```

```
public class Documentary1995Reducer extends Reducer<Text, IntWritable, Text, IntWritable> {

   public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOExcep*on,
InterruptedExcep*on {
      int sum = 0;

      for (IntWritable val : values) {
sum += val.get();
      }

      // Emit the total count of documentary movies in 1995
context.write(key, new IntWritable(sum));
   }
}
```

7.3 Write a MapReduce job that will count the total number of missing records where ‘genres’ are missing. Mapper class:

```
import java.io.IOExcep*on; import
org.apache.hadoop.io.IntWritable; import
org.apache.hadoop.io.Text; import
org.apache.hadoop.mapreduce.Mapper;

public class MissingGenresMapper extends Mapper<Object, Text, Text, IntWritable> {

   private final sta*c IntWritable one = new IntWritable(1);
private Text missingGenreKey = new Text("MissingGenresCount");

   public void map(Object key, Text value, Context context) throws IOExcep*on,
InterruptedExcep*on
{
```

```
      // Skip the header line        String
line = value.toString();        if
(line.startsWith("movieId")) return;

      // Split the line into fields
      String[] fields = line.split(",");

      if (fields.length == 3) {
         String genres = fields[2].trim();

         // Check if genres are missing             if
(genres.isEmpty())                                  {
context.write(missingGenreKey, one);
         }
      }
   }
}
```

Reducer class:

```
import java.io.IOExcep*on; import
org.apache.hadoop.io.IntWritable; import
org.apache.hadoop.io.Text; import
org.apache.hadoop.mapreduce.Reducer;

public class MissingGenresReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

   public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOExcep*on,
InterruptedExcep*on {
      int sum = 0;

      for (IntWritable val : values) {
sum += val.get();
```

```
        }

        // Emit the total count of records with missing genres
context.write(key, new IntWritable(sum));
    }
}
```

7.4 Write a MapReduce job to display only *tles of the movie having “Gold” anywhere in the *tle.

Mapper class:

```
import java.io.IOExcep*on; import
org.apache.hadoop.io.Text; import
org.apache.hadoop.mapreduce.Mapper;

public class GoldTitleMapper extends Mapper<Object, Text, Text, Text> {

    public void map(Object key, Text value, Context context) throws IOExcep*on,
InterruptedExcep*on
{
        // Skip the header line        String
line = value.toString();        if
(line.startsWith("movieId")) return;

        // Split the line into fields
        String[] fields = line.split(",");

        if (fields.length == 3) {
            String *tle = fields[1].trim();

            // Check if *tle contains "Gold"            if
(*tle.contains("Gold")) {                context.write(new
Text("GoldMovies"), new Text(*tle));
```

```
        }
      }
   }
}
```

Reducer class:

```
import java.io.IOExcep*on; import
org.apache.hadoop.io.Text; import
org.apache.hadoop.mapreduce.Reducer;

public class GoldTitleReducer extends Reducer<Text, Text, Text, Text> {

   public void reduce(Text key, Iterable<Text> values, Context context) throws IOExcep*on,
InterruptedExcep*on {
      for (Text value : values) {           context.write(new Text("MovieTitle"),
value);  // Emit the movie *tle
      }
   }
}
```

7.5 Write a MapReduce job that will display the count of the movies which belong to both Drama and Roman*c genre.

Mapper class:

```
import java.io.IOExcep*on; import
org.apache.hadoop.io.IntWritable; import
org.apache.hadoop.io.Text; import
org.apache.hadoop.mapreduce.Mapper;

public class DramaRoman*cMapper extends Mapper<Object, Text, Text, IntWritable> {

   private final sta*c IntWritable one = new IntWritable(1);
private Text dramaRoman*cKey = new Text("Drama_Roman*c");
```

```
public void map(Object key, Text value, Context context) throws
IOExcep*on, InterruptedExcep*on
{
      // Skip the header line        String
line = value.toString();        if
(line.startsWith("movieId")) return;

      // Split the line into fields
      String[] fields = line.split(",");

      if (fields.length == 3) {
         String genres = fields[2].trim();

         // Check if genres contain both "Drama" and "Roman*c"
if (genres.contains("Drama") && genres.contains("Romance")) {
context.write(dramaRoman*cKey, one);
         }
      }
   }
}
```

Reducer class:

```
import java.io.IOExcep*on; import
org.apache.hadoop.io.IntWritable; import
org.apache.hadoop.io.Text; import
org.apache.hadoop.mapreduce.Reducer;

public class DramaRoman*cReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

   public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOExcep*on,
InterruptedExcep*on {
      int sum = 0;
```

```
        for (IntWritable val : values) {
            sum += val.get();
        }

        // Emit the total count of drama and roman*c movies
context.write(key, new IntWritable(sum));
    }
}
```