PROGRAM TO JOIN TWO DATASETS latest_bitcoin and latest_crypto

DRIVER CODE OR JOB CODE TO RUN TWO MAPPERS AND REDUCER

```
        import java.io.File;
        import java.io.IOException;
        import org.apache.commons.io.FileUtils;
        import org.apache.hadoop.conf.Configuration;
        import org.apache.hadoop.fs.Path;
        import org.apache.hadoop.io.Text;
        import org.apache.hadoop.mapreduce.Job;
        import org.apache.hadoop.mapreduce.lib.input.MultipleInputs;
        import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
        import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
        import com.hadoop.design.summarization.blog.ConfigurationFactory;

        public class DriverStructuredToHierarchical {

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


        /* set the hadoop system parameter */

        System.setProperty("hadoop.home.dir", “home/gdigumu“);

        if (args.length != 3) {
        System.err.println("Please specify the input and output path");
        System.exit(-1);
        }

        Configuration conf = ConfigurationFactory.getInstance();
        Job sampleJob = Job.getInstance(conf);
        sampleJob.setJarByClass(DriverStructuredToHierarchical.class);
        TextOutputFormat.setOutputPath(sampleJob, new Path(args[2]));
        sampleJob.setOutputKeyClass(Text.class);
        sampleJob.setOutputValueClass(Text.class);
        sampleJob.setReducerClass(CryptoBitcoinJoinReducer.class);
        MultipleInputs.addInputPath(sampleJob, new Path(args[0]), TextInputFormat.class,
        BitcoinDataMapper.class);
        MultipleInputs.addInputPath(sampleJob, new Path(args[1]), TextInputFormat.class,
cryptoDataMapper.class);
        sampleJob.getConfiguration().set("validCount", "1");
        sampleJob.getConfiguration().set("totalCount", "1");
        @SuppressWarnings("unused")
        int code = sampleJob.waitForCompletion(true) ? 0 : 1;
```

```
	}
	}
```

MAPPER CODE 1

```
	import java.io.IOException;
	import org.apache.hadoop.io.Text;
	import org.apache.hadoop.mapreduce.Mapper;

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

	private Text outkey = new Text();
	private Text outvalue = new Text();
	public static final String COMMA = ",";


	public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {

	String[] values = value.toString().split(",", -1);
	String neid = values[5];
	String portid = values[6];
	outkey.set(id + location);
	outvalue.set("H" + values[4] + COMMA + values[5] + COMMA + values[6] + COMMA
	+ values[7]+COMMA+values[8]);
	context.write(outkey, outvalue);

	}
	}
```

MAPPER CODE 2

```
        import java.io.IOException;
        import org.apache.hadoop.io.Text;
        import org.apache.hadoop.mapreduce.Mapper;

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

        private Text outkey = new Text();
        private Text outvalue = new Text();
        public static final String COMMA = ",";


        public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {

        String data = value.toString();
        String[] field = data.split(",", -1);
        if (null != field && field.length > 62) {

        String neid = field[2];
        String portid = field[3];
        outkey.set(id + location);
        outvalue.set("D" + field[0] + COMMA + field[5] + COMMA + field[7].toString() + COMMA
+ field[62] + COMMA
        + field[63]);
        context.write(outkey, outvalue);

        }

        }

        }
```

REDUCER CODE

```
        import java.io.IOException;
        import java.text.ParseException;
        import java.util.ArrayList;
        import java.util.HashMap;
        import java.util.Map;
        import org.apache.hadoop.io.NullWritable;
        import org.apache.hadoop.io.Text;
        import org.apache.hadoop.mapreduce.Reducer;

        public class CryptoBitcoinJoinReducer extends Reducer<Text, Text,
NullWritable, Text> {

        private ArrayList<Text> listH = new ArrayList<Text>();
        private ArrayList<Text> listD = new ArrayList<Text>();


        public void reduce(Text key, Iterable<Text> values, Context context) throws
IOException, InterruptedException {

        listH.clear();
        listD.clear();

        for (Text text : values) {

        if (text.charAt(0) == 'H') {
        listH.add(new Text(text.toString().substring(1)));
        } else if (text.charAt(0) == 'D') {
        listD.add(new Text(text.toString().substring(1)));
        }

        }
        try {
        executeConversionLogic(context);
        } catch (ParseException e) {

        throw new IOException("Its a parse exception wrapped in IOException " +
e.getMessage());

        }

        }

        private void executeConversionLogic(Context context) throws IOException,
InterruptedException, ParseException {

        if (!listH.isEmpty() && !listD.isEmpty()) {

        for (Text hlogText : listH) {
        String[] hlog = hlogText.toString().split(",");

        for (Text dslText : listD) {

        String[] dsl = dslText.toString().split(",", -1);
        Map<String, String> maps = new HashMap<String, String>();
        maps.put("id", dsl[2]);
        maps.put("location", dsl[3]);
        JsonBuilder jsonBuilder = new JsonBuilder();
        String json = jsonBuilder.buildJson(hlog[1] + "_" + hlog[2], maps);
        context.write(NullWritable.get(), new Text(json));
        break;

        }

        }
        }
        }

        }
```