```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// AirlinePerformanceParser.java
// 공통 클래스입니다.
// 항공 통계 데이터는 콤마단위로 데이터가 저장돼 있어서 매퍼에서는 이를 구분해서 처리해야 합니다.
// 각 레코드에서는 항공 출발 지연시간과 도착 지연시간을 추출해야 합니다.
import org.apache.hadoop.io.Text;

public class AirlinePerformanceParser {
        private int year;
        private int month;
        private int arriveDelayTime = 0;
        private int departureDelayTime = 0;
        private int distance = 0;
        private boolean arriveDelayAvailable = true;
        private boolean departDelayAvailable = true;
        private boolean distanceAvailable = true;
        private String uniqueCarrier;

        public AirlinePerformanceParser(Text text) {
                try {
                        String[] colums = text.toString().split(",");

                        // 운항 연도 설정
                        year = Integer.parseInt(colums[0]);
                        // 운항 월 설정
                        month = Integer.parseInt(colums[1]);
                        // 항공사 코드 설정
                        uniqueCarrier = colums[8];

                        ////////////////////////// 지연 시간 ////////////////////////////
                        // 출발 지연 시간과 도착 지연 시간에는 NA가 값으로 들어 있는 경우가 있습니다.
                        // 이를 정숫값으로 변환하면 NumberFormatException이 발생하기 때문에
                        // NA값인지 확인하는 조건을 추가해야 합니다.
                        // NA값이 발견되면 해당 플래그를 false로 설정합니다.
                        ////////////////////////// 지연 시간 ////////////////////////////
                        // 항공사 출발 지연 시간 설정
                        if(!colums[15].equals("NA"))
                                departureDelayTime = Integer.parseInt(colums[15]);
                        else
                                departDelayAvailable = false;
                        // 항공기 도착 지연 시간 설정
                        if(!colums[15].equals("NA"))
                                arriveDelayTime = Integer.parseInt(colums[14]);
                        else
                                arriveDelayAvailable = false;
                } catch (Exception e) {
                        System.out.println("Error parsing a record :" + e.getMessage());
                }
        }

        public int getYear() { return year; }
        public int getMonth() { return month; }
        public int getArriveDelayTime() { return arriveDelayTime; }
        public int getDepartureDelayTime() { return departureDelayTime; }
        public boolean isarriveDelayAvailable() { return arriveDelayAvailable;}
        public boolean isdepartDelayAvailable() { return departDelayAvailable; }
        public String getUniqueCarrier() { return uniqueCarrier;}
        public int getDistance() { return distance;}
        public boolean isDistanceAvailable() { return distanceAvailable;}
}
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// TaggedKey
// 복합키를 구현합니다.
// 리듀스 사이드 조인의 복합키는 조인키와 태그의 조합으로 구성합니다.
// 조인키는 항공사 코드를 사용하며, 태그는 정수형 변수를 선언합니다.
// 항공사 코드를 처리하는 매퍼에서는 태그값을 0으로 설정하고, 운항 통계 데이터를 처리하는 매퍼는 태그값을 1로 설정합니다.
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.io.WritableUtils;
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
// 복합키는 WritableComparable 인터페이스를 구현합니다.
public class TaggedKey implements WritableComparable<TaggedKey> {

  private String carrierCode;  // 항공사 코드
  private Integer tag;  // 조인 테그

  // 매퍼가 태그값을 쉽게 설정할 수 있게 생성자에서 조인키와 태그값을 전달받게 합니다.
  public TaggedKey() {}
  public TaggedKey(String carrierCode, int tag) {
    this.carrierCode = carrierCode;
    this.tag = tag;
  }
  public String getCarrierCode() {
    return carrierCode;
  }
  public Integer getTag() {
    return tag;
  }
  public void setCarrierCode(String carrierCode) {
    this.carrierCode = carrierCode;
  }
  public void setTag(Integer tag) {
    this.tag = tag;
  }
  @Override
  public int compareTo(TaggedKey key) {
    int result = this.carrierCode.compareTo(key.carrierCode);
    if (result == 0) {
      return  this.tag.compareTo(key.tag);
    }
    return result;
  }
  @Override
  public void write(DataOutput out) throws IOException {
    WritableUtils.writeString(out, carrierCode);
    out.writeInt(tag);
  }
  @Override
  public void readFields(DataInput in) throws IOException {
    carrierCode = WritableUtils.readString(in);
    tag = in.readInt();
  }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// TaggedKeyComparator
// 복합키 비교기를 구현합니다.
// TaggedKey로 생성된 레코드의 정렬을 보장하려면 WritableComparator를 구현한 비교기를 구현해야 합니다.
// 복합키 비교기인 TaggedKeyComparator의 레코드 순서를 비교합니다.
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.io.WritableComparator;

public class TaggedKeyComparator extends WritableComparator {
  protected TaggedKeyComparator() {
    super(TaggedKey.class, true);
  }
  @SuppressWarnings("rawtypes")
  @Override
  public int compare(WritableComparable w1, WritableComparable w2) {
    TaggedKey k1 = (TaggedKey) w1;
    TaggedKey k2 = (TaggedKey) w2;

    // TaggedKeyComparator는 먼저 항공사 코드를 기준으로 데이터를 비교합니다.
    int cmp = k1.getCarrierCode().compareTo(k2.getCarrierCode());
    if (cmp != 0) {
      return cmp;
    }

    // 만약 항공사 코드가 같다면 태그값을 비교해 순서를 결정합니다.
    return k1.getTag().compareTo(k2.getTag());
  }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// TaggedGroupKeyPartitioner
// 그룹키 파티셔너를 구현합니다.
// 각 매퍼는 두 종류의 데이터를 출력합니다.
// 1) 출력키 : 복합키(TaggedKey), 출력값 : 항공사 이름
// 2) 출력키 : 복합키(TaggedKey), 출력값 : 운항 통계 레코드
// 각 데이터는 동일한 복합키를 처리하는 리듀서로 전달돼야 조인이 정상적으로 처리됩니다.
// 이를 위해서 그룹키 파티셔너를 구현합니다.
// 그룹키 파티셔너는 org.apache.hadoop.mapreduce.Partitioner를 상속받으며,
// 파티션 번호는 항공사 코드의 해시값을 이용해 계산합니다.
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Partitioner;

public class TaggedGroupKeyPartitioner extends Partitioner<TaggedKey, Text> {
  @Override
  public int getPartition(TaggedKey key, Text val, int numPartitions) {

    // 항공사 코드의 해시값으로 파티션 계산
    int hash = key.getCarrierCode().hashCode();
    int partition = hash % numPartitions;

    return partition;
  }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// TaggedGroupKeyComparator
// 그룹키 비교기를 구현합니다.
// TaggedGroupKeyPartitioner을 적용했을 때, 리듀서는 동일한 항공사 코드의 데이터를 내려받을 수 있지만,
// 이 데이터들은 순서가 보장되지 않으므로, 그룹키 비교기를 구현해 데이터를 정렬해야 합니다.
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.io.WritableComparator;

public class TaggedGroupKeyComparator extends WritableComparator {
  protected TaggedGroupKeyComparator() {
    super(TaggedKey.class, true);
  }
  @SuppressWarnings("rawtypes")
  @Override
  public int compare(WritableComparable w1, WritableComparable w2) {
    TaggedKey k1 = (TaggedKey) w1;
    TaggedKey k2 = (TaggedKey) w2;

    // 그룹키인 항공사 코드값 정렬
    return k1.getCarrierCode().compareTo(k2.getCarrierCode());
  }
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CarrierCodeMapper
// 항공사 코드 데이터를 태깅하는 매퍼를 구현합니다.
// 항공사 코드 데이터를 입력 데이터로 전달받으며,
// 출력키는 “항공사 코드, 태그(0)”, 출력값은 “해당 항공사 코드의 항공사 이름”을 출력합니다.
import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class CarrierCodeMapper extends Mapper<LongWritable, Text, TaggedKey, Text> {
  // 출력 키, 출력 값
  TaggedKey outputKey = new TaggedKey();
  Text outValue = new Text();

  public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
    // CarrierCodeParser를 이용해 콤마(,)로 구성된 입력 데이터를 처리합니다.
    // 출력키인 TaggedKey에 항공사 코드와 태그값을 0으로 설정합니다.
    // 출력키인 “항공사 코드, 태그(0)”를 설정하고, 출력값에는 항공사 이름을 설정합니다.
    // 출력값에 항공사 이름을 설정한 후, 출력 데이터를 생성합니다.
    CarrierCodeParser parser = new CarrierCodeParser(value);
    outputKey.setCarrierCode(parser.getCarrierCode());
    outputKey.setTag(0);
    outValue.set(parser.getCarrierName());
    context.write(outputKey, outValue);
  }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// MapperWithReduceSideJoin
// 항공기 운항 통계 데이터 매퍼를 구현합니다.
// 항공 운항 통계 데이터를 입력 데이터로 전달받습니다.
// 출력키는 TaggedKey, 출력값은 입력 데이터의 값을 그대로 출력합니다.
import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class MapperWithReduceSideJoin extends Mapper<LongWritable, Text, TaggedKey, Text> {
  TaggedKey outputKey = new TaggedKey();  // 출력 키 설정

  // 맵 메서드에서는 AirlinePerformanceParser를 이용해 운항 통계 정보를 조회합니다.
  public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
    AirlinePerformanceParser parser = new AirlinePerformanceParser(value);

    // 출력키인 TaggedKey의 항공사 코드와 태그값을 설정하며, 이 때 태그값은 1을 사용합니다.
    outputKey.setCarrierCode(parser.getUniqueCarrier());
    outputKey.setTag(1);

    // 출력값은 입력 레코드를 그대로 설정한 후, 출력 데이터를 생성합니다.
    context.write(outputKey, value);
  }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// ReducerWithReduceSideJoin
// 조인 리듀서를 구현합니다.
// 입력키는 TaggedKey이며, 입력키는 Text 타입의 항공사 이름과 운항 통계 레코드로 구성됩니다.
// 각 데이터 목록의 첫 번째 레코드는 반드시 태그값이 0인, 항공사 이름이 전달됩니다.
// 위와 같은 정렬 순서가 보장되는 이유는 TaggedGroupKeyComparator를 적용했기 때문입니다.
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class ReducerWithReduceSideJoin extends Reducer<TaggedKey, Text, Text, Text> {
  private Text outputKey = new Text();  // 출력키
  private Text outputValue = new Text();  // 출력값

  public void reduce(TaggedKey key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
    // 입력 값을 순회하기 위해 Iterator 객체를 생성합니다.
    Iterator<Text> iterator = values.iterator();

    // 첫 번째 레코드에 있는 항공사 이름을 조회합니다.
    Text carrierName = new Text(iterator.next());

    // 나머지 레코드를 순회하면서 출력 데이터를 생성합니다.
    // 이 때 출력키는 항공사 코드를 사용하고,
    //       출력값은 첫 번째 레코드에서 조회한 항공사 이름과 통계 레코드를 조합해서 설정합니다.
    while (iterator.hasNext()) {
      Text record = iterator.next();
      outputKey.set(key.getCarrierCode());
      outputValue = new Text(carrierName.toString() + "₩t" + record.toString());
      context.write(outputKey, outputValue);
    }
  }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// ReduceSideJoin
// 드라이버 클래스를 구현합니다.
// 사용자가 입력한 조인할 두 개의 데이터를 MultipleInputs의 static메서드인 addInputPath를 호출해 입력 데이터에 추가합니다.
// 이 때 입력 데이터 타입과 매퍼도 함께 설정합니다.
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
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.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
public class ReduceSideJoin extends Configured implements Tool {
  public static void main(String[] args) throws Exception {
    // Tool 인터페이스 실행
    int res = ToolRunner.run(new Configuration(), new ReduceSideJoin(), args);
    System.out.println("MR-Job Result:" + res);
  }
  public int run(String[] args) throws Exception {
    String[] otherArgs = new GenericOptionsParser(getConf(), args).getRemainingArgs();
    // 입출력 데이터 경로 확인
    if (otherArgs.length != 3) {
      System.err.println("Usage: ReduceSideJoin <metadata> <in> <out>");
      System.exit(2);
    }
    Job job = new Job(getConf(), "ReduceSideJoin");                    // Job 이름 설정
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[2]));      // 출력 데이터 경로 설정

    // Job 클래스 설정
    job.setJarByClass(ReduceSideJoin.class);
    job.setPartitionerClass(TaggedGroupKeyPartitioner.class);
    job.setGroupingComparatorClass(TaggedGroupKeyComparator.class);
    job.setSortComparatorClass(TaggedKeyComparator.class);

    // Reducer 클래스 설정
    job.setReducerClass(ReducerWithReduceSideJoin.class);
    job.setMapOutputKeyClass(TaggedKey.class);
    job.setMapOutputValueClass(Text.class);

    // 입출력 데이터 포맷 설정
    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    // 출력키 및 출력값 유형 설정
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);

    // MultipleInputs 설정
    MultipleInputs.addInputPath(job, new Path(otherArgs[0]), TextInputFormat.class, CarrierCodeMapper.class);
    MultipleInputs.addInputPath(job, new Path(otherArgs[1]), TextInputFormat.class, MapperWithReduceSideJoin.class);
    job.waitForCompletion(true);
    return 0;
  }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CarrierCodeParser.java
// 항공사 코드 데이터를 읽는 코드를 구현합니다.

import org.apache.hadoop.io.Text;
public class CarrierCodeParser {
  private String carrierCode;  // 항공사 코드
  private String carrierName;  // 항공사 이름

  public CarrierCodeParser(Text value) {
    this(value.toString());
  }
  public CarrierCodeParser(String value) {
    try {
      String[] colums = value.split(",");
      if (colums != null && colums.length > 0) {
        carrierCode = colums[0];
        carrierName = colums[1];
      }
    } catch (Exception e) {
      System.out.println("Error parsing a record :" + e.getMessage());
    }
  }
  public String getCarrierCode() {
    return carrierCode;
  }
  public String getCarrierName() {
    return carrierName;
  }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```