```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// 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;}
}
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// MapperWithMapSideJoin.java
// 조인 연산에서 사용할 매퍼를 구현합니다.
// 처음 매퍼가 생성될 때 분산 캐시에서 항공사 코드 데이터를 조회해 내부 변수에 저장한 후,
// 맵 매서드가 실행될 때 각 레코드에 해당하는 항공사 이름을 키값으로 하는 출력 데이터를 생성합니다.
import org.apache.hadoop.filecache.DistributedCache;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.Hashtable;

public class MapperWithMapSideJoin extends Mapper<LongWritable, Text, Text, Text> {
  // 분산 캐시에서 조회한 항공사 코드 데이터를 저장할 HashMap 객체를 선언합니다.
  private Hashtable<String, String> joinMap = new Hashtable<String, String>();

  // map 출력키
  private Text outputKey = new Text();
  @Override
  public void setup(Context context) throws IOException, InterruptedException {
    try {
      // 매퍼가 분산 캐시를 불필요하게 접근하는 것을 줄이도록 이 setup 메서드에서만 분산 캐시를 조회합니다.
      // 분산 캐시에 저장된 텍스트 파일을 조회하기 때문에 DistributedCache의 getLocalCacheFiles 메서드를 호출해
      // 분산 캐시에 있는 전체 로컬 파일을 조회합니다.
      Path[] cacheFiles = DistributedCache.getLocalCacheFiles(context.getConfiguration());

      // 캐시 파일이 정상적으로 반환되면 전체 파일에서 첫 번째 파일을 이용해 BufferedReader를 생성합니다.
      if (cacheFiles != null && cacheFiles.length > 0) {
        String line;
        BufferedReader br = new BufferedReader(new FileReader(cacheFiles[0].toString()));

        try {
          // BufferedReader를 순회하면서 CarrierCodeParser를 이용해 항공사 코드와 항공사 이름을 조회합니다.
          // 정보를 HashMap 객체에 등록합니다.
          while ((line = br.readLine()) != null) {
            CarrierCodeParser codeParser = new CarrierCodeParser(line);
            joinMap.put(codeParser.getCarrierCode(), codeParser.getCarrierName());
          }
        } finally {
          br.close();
        }
      } else {
        System.out.println("cache files is null!");
      }
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

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

    // HashMap 객체에서 해당 항공사 이름을 조회해 데이터를 출력합니다.
    AirlinePerformanceParser parser = new AirlinePerformanceParser(value);
    outputKey.set(parser.getUniqueCarrier());
    context.write(outputKey, new Text(joinMap.get(parser.getUniqueCarrier()) + "₩t" + value.toString()));
  }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// MapSideJoin.java
// 맵 사이드 조인을 실행하는 드라이버 클래스를 구현합니다.
// 다른 드라이버 클래스와 마찬가지로 Configuration을 상속받고 Tool 인터페이스를 선언해 run 메서드에 잡에 대한 로직을 구현했습니다.
// 잡 객체를 선언하기 전에 Configuration에 분산캐시를 설정해야 합니다.
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.filecache.DistributedCache;
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.FileInputFormat;
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 MapSideJoin extends Configured implements Tool {          // Tool 인터페이스 실행
  public static void main(String[] args) throws Exception {
    int res = ToolRunner.run(new Configuration(), new MapSideJoin(), 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 (파라미터가 3개인지)확인
    if (otherArgs.length != 3) {
      System.err.println("Usage: MapSideJoin <metadata> <in> <out>");
      System.exit(2);
    }

    Job job = new Job(getConf(), "MapSideJoin");      // Job 이름 설정

    // DistributedCache의 addCacheFile메서드를 호출해 로컬 캐시 파일을 추가합니다.
    DistributedCache.addCacheFile(new Path(otherArgs[0]).toUri(), job.getConfiguration()); // 분산 캐시 설정

    // 입출력 데이터 경로 설정
    FileInputFormat.addInputPath(job, new Path(otherArgs[1]));
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[2]));

    job.setJarByClass(MapSideJoin.class);                   // Job 클래스 설정
    job.setMapperClass(MapperWithMapSideJoin.class);    // Mapper 설정

    // Reducer를 구현하지 않았으므로 잡에 대한 리듀스 태스크 개수는 0
    job.setNumReduceTasks(0);                               // Reducer 설정

    // 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);

    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;
  }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```