```
package com.marcolussetti.opendotamatchescondenser;

import com.jsoniter.JsonIterator;
import com.jsoniter.any.Any;
import com.jsoniter.output.JsonStream;
import gnu.trove.map.hash.THashMap;
import gnu.trove.set.hash.THashSet;
import org.simpleflatmapper.csv.CsvParser;
import picocli.CommandLine;
import picocli.CommandLine.Option;
import picocli.CommandLine.Command;
import picocli.CommandLine.Parameters;

import java.io.*;
import java.time.*;
import java.time.format.DateTimeFormatter;
import java.util.*;
import java.util.concurrent.Callable;
import java.util.concurrent.TimeUnit;

@Command(description = "Process OpenDota Matches File",
        name = "processopendota",
        mixinStandardHelpOptions = true,
        version = "processopendota 0.3")
class ProcessOpenDota implements Callable<Void> {
    // ARGUMENTS
    @Option(names = {"-x", "--extract-to-json"},
            description = "Extract an existing .ser file to a JSON file.")
    private File extractToJson = null;

    @Option(names = {"-c", "--condense"},
            description = "Condense the input openDota CSV file. If file is GunZipped
 (.gz), extract it first.")
    private File condense = null;

    @Option(names = {"-o", "--only-count"},
            description = "Only count picks, do not record wins & losses")
    private Boolean onlyCount = false;

    @Parameters(paramLabel = "OUTPUT",
            description = "Output file for either extract or condense")
    private File output = null;

    // CONSTANTS
    public static final int MATCHES_NO = 1191768403;
    public static final int DAYS_NO = 1870;
    public static final int REPORT_THRESHOLD = 1000000; // Report progress every
 million rows
    public static final int SERIALIZE_THRESHOLD = 10000000; // Serialize every 10
 million rows

    // VARIABLES
    // Keep track of progress
    private LocalDateTime startOfParsing;
    private THashSet<Long> allDates = new THashSet<>();
    private int recordCounter = 0;
    // Store the data {date: Long, {hero#: int -> picks# int}}
    private THashMap<Long, THashMap<Integer, Integer[]>> data = new THashMap<>();

    private void condenseInputFile(File input, File output, boolean onlyCount) {
        this.startOfParsing = LocalDateTime.now();

        // Main loop!
        FileReader fileReader;
        try {
            fileReader = new FileReader(input);
            Iterator<String[]> csvReader = CsvParser.iterator(fileReader);
            String[] headers = csvReader.next();
            // Iterate through stuff
```

```java
            while (csvReader.hasNext()) {
                String[] row = csvReader.next();

                parseRow(row, onlyCount);

                if (recordCounter % REPORT_THRESHOLD == 0) {
                    reportProgress(this.recordCounter, this.allDates.size(), this.startOfParsing);

                    if (recordCounter % SERIALIZE_THRESHOLD == 0) {
                        String destFolder = output.getParent();
                        String[] destFile = output.getName().split("\\.");
                        File outputFile = new File(destFolder + File.separator + destFile[0] + "_" + (recordCounter / SERIALIZE_THRESHOLD) + "." + destFile[1]);

                        serializeData(data, outputFile);
                    }
                }
            }

            reportProgress(this.recordCounter, this.allDates.size(), this.startOfParsing);
            serializeData(data, output);
        } catch (IOException e) {
            e.printStackTrace();
        }

    }

    private void extractToJson(File input, File output) {
        THashMap<Long, THashMap<Integer, Integer[]>> hashMap = deserializeData(input);

        writeJSON(hashMap, output);
    }

    private void parseRow(String[] row, boolean onlyCount) {
        // Extract relevant fields
        long startTime = Long.parseLong(row[3]);
        String pgroup = row[26];
        boolean radiantWin = row[2].equals("t");

        // Parse date
        Long date = extractDate(startTime).toEpochDay();

        // Parse picks
        ArrayList<Integer[]> heroesPicked = extractHeroesPicked(pgroup, radiantWin);

        // Update copy of local map
        THashMap<Integer, Integer[]> todayPicks = this.data.getOrDefault(date, new THashMap<Integer, Integer[]>());
        heroesPicked.forEach(heroRecord -> {
            int hero = heroRecord[0];
            boolean won = heroRecord[1] == 1;

            Integer[] counts = todayPicks.getOrDefault(hero, new Integer[]{0, 0});
            if (onlyCount || won)
                counts[0] += 1;
            else
                counts[1] += 1;
            todayPicks.put(hero, counts);
        });

        // Push to global map
        this.data.put(date, todayPicks);

        // Tracking progress
        allDates.add(date);
        recordCounter++;
    }

    private static LocalDate extractDate(long epochTimeInSeconds) {
        return LocalDateTime.ofInstant(
                Instant.ofEpochSecond(epochTimeInSeconds),
                ZoneId.of("UTC")
        ).toLocalDate();
    }

    private static ArrayList<Integer[]> extractHeroesPicked(String jsonInput, boolean radiantWon) {
        ArrayList<Integer[]> heroes = new ArrayList<>();

        JsonIterator iterator = JsonIterator.parse(jsonInput);
        Map<String, Any> jsonObject = null;
        try {
            jsonObject = iterator.read(Any.class).asMap();
        } catch (IOException e) {
            e.printStackTrace();
        }

        jsonObject.forEach((index, object) -> {
            int heroId = object.get("hero_id").toInt();
            boolean isRadiant = object.get("player_slot").toInt() <= 127;
            Integer[] heroRecord = {heroId, ((isRadiant && radiantWon) || (!isRadiant && !radiantWon)) ? 1 : 0 };
            heroes.add(heroRecord);
        });

        return heroes;

    }

    private static void reportProgress(int recordCounter, int days, LocalDateTime startOfParsing) {
        Duration elapsed = Duration.between(startOfParsing, LocalDateTime.now());
        long elapsedMillis = elapsed.toMillis();
        DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy/MM/dd HH:mm:ss");
        double rowsPerSec = (double) recordCounter / elapsedMillis * 1000;

        System.out.printf(
                "\n%s (%s elapsed - %s remaining) | %9.2f rows/s | %,6.2f million rows (%6.2f%%)| %4d days (%6.2f%%)",
                dtf.format(LocalDateTime.now()),                                        // current time
                formatTimeDifference(elapsedMillis),                                    // elapsed time
                formatTimeDifference((long) ((MATCHES_NO - recordCounter) / rowsPerSec * 1000)),// remaining time (est.)
                (double) recordCounter / elapsedMillis * 1000,                          // rows per second
                (double) recordCounter / 1000000,                                       // rows processed (mils)
                (double) recordCounter / MATCHES_NO * 100,                              // % of rows processed
                days,                                                                   // days tracked
                days / (float) DAYS_NO * 100                                            // % of days tracked
        );
    }

    private static String formatTimeDifference(long millis) {
        // From https://stackoverflow.com/a/44142896/6238740
        return String.format("%02d:%02d:%02d",
                TimeUnit.MILLISECONDS.toHours(millis),
                TimeUnit.MILLISECONDS.toMinutes(millis) -
                        TimeUnit.HOURS.toMinutes(TimeUnit.MILLISECONDS.toHours(
millis)),
                TimeUnit.MILLISECONDS.toSeconds(millis) -
                        TimeUnit.MINUTES.toSeconds(TimeUnit.MILLISECONDS.toMinutes(millis)));
    }

    private static void serializeData(THashMap<Long, THashMap<Integer, Integer[]>> data, File output) {

        // From https://beginnersbook.com/2013/12/how-to-serialize-hashmap-in-java/
        FileOutputStream fos = null;
        try {
            fos = new FileOutputStream(output);
            ObjectOutputStream oos = new ObjectOutputStream(fos);
            oos.writeObject(data);
            oos.close();
            fos.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        System.out.print("\n> Saved data to " + output.getAbsolutePath());
    }

    public static THashMap<Long, THashMap<Integer, Integer[]>> deserializeData(File file) {
        // From https://beginnersbook.com/2013/12/how-to-serialize-hashmap-in-java/
        THashMap<Long, THashMap<Integer, Integer[]>> hashMap;
        try {
            FileInputStream fis = new FileInputStream(file);
            ObjectInputStream ois = new ObjectInputStream(fis);
            hashMap = (THashMap<Long, THashMap<Integer, Integer[]>>) ois.readObject();
            ois.close();
            fis.close();
        } catch (IOException ioe) {
            ioe.printStackTrace();
            return null;
        } catch (ClassNotFoundException c) {
            System.out.println("Class not found");
            c.printStackTrace();
            return null;
        }
        return hashMap;
    }

    public static void writeJSON(THashMap<Long, THashMap<Integer, Integer[]>> hashMap, File outputFile) {

        String output = JsonStream.serialize(hashMap);
        try {
            outputFile.createNewFile();
        } catch (IOException e) {
            e.printStackTrace();
        }

        try (PrintStream out = new PrintStream(new FileOutputStream(outputFile))) {
            out.print(output);
            out.flush();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }

    }

    public static void main(String[] args) {
        CommandLine.call(new ProcessOpenDota(), args);
    }

    @Override
    public Void call() throws Exception {
        // BUSINESS LOGIC

        if (extractToJson == null && condense == null) {
            System.out.println("Well you need to select something... try --help");
            return null;
        }
        if (extractToJson != null && condense != null) {
            System.out.println("Can't have it both ways... try --help");
            return null;
        }
        if (output == null) {
            System.out.println("Must provide an output file!");
            return null;
        }

        if (extractToJson != null) {
            System.out.println("Converting from SER to JSON");
            extractToJson(extractToJson, output);
            System.out.println("Conversion complete: " + output.getAbsolutePath());
        }
        if (condense != null) {
            System.out.println("Condensing from CSV to SER");
            condenseInputFile(condense, output, onlyCount);
            System.out.println("Condensing complete: " + output.getAbsolutePath());
        }

        return null;
    }
}

```