```java
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);
```

```
65              String[] headers = csvReader.next();
66              // Iterate through stuff
67              while (csvReader.hasNext()) {
68                  String[] row = csvReader.next();
69
70                  parseRow(row, onlyCount);
71
72                  if (recordCounter % REPORT_THRESHOLD == 0) {
73                      reportProgress(this.recordCounter, this.allDates.size(), this.
    startOfParsing);
74
75                      if (recordCounter % SERIALIZE_THRESHOLD == 0) {
76                          String destFolder = output.getParent();
77                          String[] destFile = output.getName().split("\\.");
78                          File outputFile = new File(
79                                  destFolder + File.separator + destFile[0] +"_" + (
80                                          recordCounter / SERIALIZE_THRESHOLD
81                                  ) + "." + destFile[1]);
82
83                          serializeData(data, outputFile);
84                      }
85                  }
86              }
87
88              reportProgress(this.recordCounter, this.allDates.size(), this.
    startOfParsing);
89              serializeData(data, output);
90          } catch (IOException e) {
91              e.printStackTrace();
92          }
93
94      }
95
96      private void extractToJson(File input, File output) {
97          THashMap<Long, THashMap<Integer, Integer[]>> hashMap = deserializeData(input
    );
98
99          writeJSON(hashMap, output);
100     }
101
102     private void parseRow(String[] row, boolean onlyCount) {
103         // Extract relevant fields
104         long startTime = Long.parseLong(row[3]);
105         String pgroup = row[26];
106         boolean radiantWin = row[2].equals("t");
107
108         // Parse date
109         Long date = extractDate(startTime).toEpochDay();
110
111         // Parse picks
112         ArrayList<Integer[]> heroesPicked = extractHeroesPicked(pgroup, radiantWin);
113
114         // Update copy of local map
115         THashMap<Integer, Integer[]> todayPicks = this.data.getOrDefault(date, new
    THashMap<Integer, Integer[]>());
116         heroesPicked.forEach(heroRecord -> {
117             int hero = heroRecord[0];
118             boolean won = heroRecord[1] == 1;
119
120             Integer[] counts = todayPicks.getOrDefault(hero, new Integer[]{0, 0});
121             if (onlyCount || won)
122                 counts[0] += 1;
123             else
124                 counts[1] += 1;
125             todayPicks.put(hero, counts);
126         });
127
```

```
128         // Push to global map
129         this.data.put(date, todayPicks);
130
131         // Tracking progress
132         allDates.add(date);
133         recordCounter++;
134     }
135
136     private static LocalDate extractDate(long epochTimeInSeconds) {
137         return LocalDateTime.ofInstant(
138                 Instant.ofEpochSecond(epochTimeInSeconds),
139                 ZoneId.of("UTC")
140         ).toLocalDate();
141     }
142
143     private static ArrayList<Integer[]> extractHeroesPicked(String jsonInput,
    boolean radiantWon) {
144         ArrayList<Integer[]> heroes = new ArrayList<>();
145
146         JsonIterator iterator = JsonIterator.parse(jsonInput);
147         Map<String, Any> jsonObject = null;
148         try {
149             jsonObject = iterator.read(Any.class).asMap();
150         } catch (IOException e) {
151             e.printStackTrace();
152         }
153
154         jsonObject.forEach((index, object) -> {
155             int heroId = object.get("hero_id").toInt();
156             boolean isRadiant = object.get("player_slot").toInt() <= 127;
157             Integer[] heroRecord = {heroId, ((isRadiant && radiantWon) || (!
    isRadiant && !radiantWon)) ? 1 : 0 };
158             heroes.add(heroRecord);
159         });
160
161         return heroes;
162
163     }
164
165     private static void reportProgress(int recordCounter, int days, LocalDateTime
    startOfParsing) {
166         Duration elapsed = Duration.between(startOfParsing, LocalDateTime.now());
167         long elapsedMillis = elapsed.toMillis();
168         DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy/MM/dd HH:mm:ss");
169         double rowsPerSec = (double) recordCounter / elapsedMillis * 1000;
170
171         System.out.printf(
172                 "\n%s (%s elapsed - %s remaining) | %9.2f rows/s | %,6.2f million
    rows (%6.2f%%)| %4d days (%6.2f%%)",
173                 dtf.format(LocalDateTime.now()),
                                                    // current time
174                 formatTimeDifference(elapsedMillis),
                                                  // elapsed time
175                 formatTimeDifference((long) ((MATCHES_NO - recordCounter) /
    rowsPerSec * 1000)),// remaining time (est.)
176                 (double) recordCounter / elapsedMillis * 1000,
                                       // rows per second
177                 (double) recordCounter / 1000000,
                                                   // rows processed (mils)
178                 (double) recordCounter / MATCHES_NO * 100,
                                      // % of rows processed
179                 days,
                                                                          // days
    tracked
180                 days / (float) DAYS_NO * 100
                                                        // % of days tracked
181         );
```

```
182     }
183
184     private static String formatTimeDifference(long millis) {
185         // From https://stackoverflow.com/a/44142896/6238740
186         return String.format("%02d:%02d:%02d",
187                 TimeUnit.MILLISECONDS.toHours(millis),
188                 TimeUnit.MILLISECONDS.toMinutes(millis) -
189                         TimeUnit.HOURS.toMinutes(TimeUnit.MILLISECONDS.toHours(
    millis)),
190                 TimeUnit.MILLISECONDS.toSeconds(millis) -
191                         TimeUnit.MINUTES.toSeconds(TimeUnit.MILLISECONDS.toMinutes(
    millis)));
192     }
193
194     private static void serializeData(THashMap<Long, THashMap<Integer, Integer[]>>
    data, File output) {
195
196         // From https://beginnersbook.com/2013/12/how-to-serialize-hashmap-in-java/
197         FileOutputStream fos = null;
198         try {
199             fos = new FileOutputStream(output);
200             ObjectOutputStream oos = new ObjectOutputStream(fos);
201             oos.writeObject(data);
202             oos.close();
203             fos.close();
204         } catch (IOException e) {
205             e.printStackTrace();
206         }
207         System.out.print("\n> Saved data to " + output.getAbsolutePath());
208     }
209
210     public static THashMap<Long, THashMap<Integer, Integer[]>> deserializeData(File
    file) {
211         // From https://beginnersbook.com/2013/12/how-to-serialize-hashmap-in-java/
212         THashMap<Long, THashMap<Integer, Integer[]>> hashMap;
213         try {
214             FileInputStream fis = new FileInputStream(file);
215             ObjectInputStream ois = new ObjectInputStream(fis);
216             hashMap = (THashMap<Long, THashMap<Integer, Integer[]>>) ois.readObject(
    );
217             ois.close();
218             fis.close();
219         } catch (IOException ioe) {
220             ioe.printStackTrace();
221             return null;
222         } catch (ClassNotFoundException c) {
223             System.out.println("Class not found");
224             c.printStackTrace();
225             return null;
226         }
227         return hashMap;
228     }
229
230     public static void writeJSON(THashMap<Long, THashMap<Integer, Integer[]>>
    hashMap, File outputFile) {
231
232         String output = JsonStream.serialize(hashMap);
233         try {
234             outputFile.createNewFile();
235         } catch (IOException e) {
236             e.printStackTrace();
237         }
238
239         try (PrintStream out = new PrintStream(new FileOutputStream(outputFile))) {
240             out.print(output);
241             out.flush();
242         } catch (FileNotFoundException e) {
```

```
243             e.printStackTrace();
244         }
245
246     }
247
248     public static void main(String[] args) {
249         CommandLine.call(new ProcessOpenDota(), args);
250     }
251
252     @Override
253     public Void call() throws Exception {
254         // BUSINESS LOGIC
255
256         if (extractToJson == null && condense == null) {
257             System.out.println("Well you need to select something... try --help");
258             return null;
259         }
260         if (extractToJson != null && condense != null) {
261             System.out.println("Can't have it both ways... try --help");
262             return null;
263         }
264         if (output == null) {
265             System.out.println("Must provide an output file!");
266             return null;
267         }
268
269         if (extractToJson != null) {
270             System.out.println("Converting from SER to JSON");
271             extractToJson(extractToJson, output);
272             System.out.println("Conversion complete: " + output.getAbsolutePath());
273         }
274         if (condense != null) {
275             System.out.println("Condensing from CSV to SER");
276             condenseInputFile(condense, output, onlyCount);
277             System.out.println("Condensing complete: " + output.getAbsolutePath());
278         }
279
280         return null;
281     }
282 }
283
```