Big Data Processing Course

Spring Semester 2016

Homework 1 Validation.

May 11, 2016

Chapter 1

Validation setup

1.1 Maven POM.XML

Your project were checked with the following maven configuration files, **pom.xml**. In case your project has its own dependencies you need to add them into dependencies section of the **pom.xml**.

Listing 1.1: pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>univ.bigdata.course</groupId>
  <artifactId>hw1</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>hw1</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <sourceJDK>1.8</sourceJDK>
    <targetJDK>1.8</targetJDK>
  </properties>

  <dependencies>
    <dependency>
      <groupId>net.sf.jopt-simple</groupId>
      <artifactId>jopt-simple</artifactId>
      <version>4.6</version>
    </dependency>
</dependencies>

  <build>
```

```
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.3</version>
        <configuration>
          <source>${sourceJDK}</source>
          <target>${targetJDK}</target>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-jar-plugin</artifactId>
        <version>2.6</version>
        <configuration>
          <archive>
            <manifest>
              <addClasspath>true</addClasspath>
              <mainClass>univ.bigdata.course.MoviesReviewsQueryRunner</mainClass>
            </manifest>
          </archive>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

1.2 Main program

Your submissions were checked with following main program, which very similar to the structure of the **MoviesReviewsQueryRunner.java**, in fact it has same execution queries, but divided into 4 different parts. It consist from following methods:

1. **gradingFirstPart** - first part
2. **gradingSecondPart** - second part
3. **gradingThirdPart** - third part
4. **gradingFourthPart** - fourth part

Each of these method produces different output file with suffix **_NUM.txt**, where NUM is the number of the checked part, for instance **query_out1_1.txt** is the output of the first part.

Since there are might be a variety of different implementations of **MoviesProvider** interface I will leave for you to fullfill the **initialize** method with the code which will be capable to create instance of the movies provider.

NOTE!!!

You are not allowed to change any other lines of the code and you are not expected to change or adjust files paths (values of input parameters). File paths should be exactly the same as they passed into the main program.

Listing 1.2: GradingReviewsQueryRunner.java

```
package univ.bigdata.course;

import joptsimple.OptionParser;
import joptsimple.OptionSet;
import univ.bigdata.course.providers.TestFileIOMoviesProvider;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.PrintStream;

public class GradingReviewsQueryRunner {

    public static final String INPUT_FILE = "inputFile";

    public static final String OUTPUT_FILE = "outputFile";

    private OptionSet options;

    private MoviesStorage storage;

    public GradingReviewsQueryRunner(String... args) {
        final OptionParser parser = new OptionParser();
        parser.accepts(INPUT_FILE).withRequiredArg();
        parser.accepts(OUTPUT_FILE).withRequiredArg();

        this.options = parser.parse(args);

        if (!options.has(INPUT_FILE) || !options.has(OUTPUT_FILE)) {
            System.out.println("Error, you should provide movies reviews" +
                " input text file and output file to persist queries results.");
            System.out.println("-" + INPUT_FILE + "=<path_to_input_file> -" +
                OUTPUT_FILE + "=<path_to_output_file>");
            System.exit(-1);
        }
    }

    public static void main(String[] args) throws IOException {
        final GradingReviewsQueryRunner grader = new GradingReviewsQueryRunner(args);
        grader.initialize();
        grader.grade();
    }

    private void initialize() throws IOException {
        String inputFileName = options.valueOf(INPUT_FILE).toString();
        MoviesProvider provider = ...; //TODO: Your code goes here.
        this.storage = new MoviesStorage(provider);
    }
```

```
private void grade() throws IOException {
        gradingFirstPart();
        gradingSecondPart();
        gradingThirdPart();
        gradingFourthPart();
}

private void gradingFourthPart() {
    final String file = options.valueOf(OUTPUT_FILE).toString() + "_4.txt";
    try (final PrintStream printer = new PrintStream(file)) {
        printer.println("Most popular movie with highest average score, " +
            "reviewed by at least 20 users " +
            mostPopularMovieReviewedByKUsers(storage, 20));
        printer.println("Most popular movie with highest average score, " +
            "reviewed by at least 15 users " +
            mostPopularMovieReviewedByKUsers(storage, 15));
        printer.println("Most popular movie with highest average score, " +
            "reviewed by at least 10 users " +
            mostPopularMovieReviewedByKUsers(storage, 10));
        printer.println("Most popular movie with highest average score, " +
            "reviewed by at least 5 users " +
            mostPopularMovieReviewedByKUsers(storage, 5));

        printer.println();
        printer.println("Compute top 10 most helpful users.");
        storage.topKHelpfullUsers(10)
                .entrySet()
                .forEach(pair -> printer.println("User id = [" + pair.getKey() + "
                helpfulness [" + pair.getValue() + "]."));

        printer.println();
        printer.println("Compute top 100 most helpful users.");
        storage.topKHelpfullUsers(100)
                .entrySet()
                .forEach(pair -> printer.println("User id = [" + pair.getKey() + "
                helpfulness [" + pair.getValue() + "]."));

        printer.println();
        printer.println("Total number of distinct movies reviewed [" +
            storage.moviesCount() + "].");
        printer.println("THE END.");
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private void gradingThirdPart() throws FileNotFoundException {
    final String file = options.valueOf(OUTPUT_FILE).toString() + "_3.txt";
    try (final PrintStream printer = new PrintStream(file)) {
        printer.println("Computing TOP100 words count");
```

```
            storage.moviesReviewWordsCount(100)
                    .entrySet()
                    .forEach(pair -> printer.println("Word = [" + pair.getKey() +
                    "], number of occurrences [" + pair.getValue() + "]."));

            printer.println();
            printer.println("Computing TOP100 words count for TOP100 movies");
            storage.topYMoviewsReviewTopXWordsCount(100, 100)
                    .entrySet()
                    .forEach(pair -> printer.println("Word = [" + pair.getKey() +
                    "], number of occurrences [" + pair.getValue() + "]."));

            printer.println("Computing TOP100 words count for TOP10 movies");
            storage.topYMoviewsReviewTopXWordsCount(10, 100)
                    .entrySet()
                    .forEach(pair -> printer.println("Word = [" + pair.getKey() +
                    "], number of occurrences [" + pair.getValue() + "]."));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private void gradingSecondPart() {
        final String file = options.valueOf(OUTPUT_FILE).toString() + "_2.txt";
        try (final PrintStream printer = new PrintStream(file)) {
            storage.reviewCountPerMovieTopKMovies(4)
                    .entrySet()
                    .stream()
                    .forEach(pair -> printer.println("Movie product id = [" +
                    pair.getKey() +
                    "], reviews count [" + pair.getValue() + "]."));

            printer.println();
            printer.println("The most reviewed movie product id is " +
                storage.mostReviewedProduct());

            printer.println();
            printer.println("Computing 90th percentile of all movies average.");
            storage.getMoviesPercentile(90).stream().forEach(printer::println);

            printer.println();
            printer.println("Computing 50th percentile of all movies average.");
            storage.getMoviesPercentile(50).stream().forEach(printer::println);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private void gradingFirstPart() {
        final String file = options.valueOf(OUTPUT_FILE).toString() + "_1.txt";
```

```
        try (final PrintStream printer = new PrintStream(file)) {
            printer.println("Getting list of total movies average.");

            printer.println();
            printer.println("TOP2.");
            storage.getTopKMoviesAverage(2).stream().forEach(printer::println);
            printer.println();
            printer.println("TOP4.");
            storage.getTopKMoviesAverage(4).stream().forEach(printer::println);

            printer.println("Total average: " + storage.totalMoviesAverageScore());

            printer.println();
            printer.println("The movie with highest average:  " +
                storage.movieWithHighestAverage());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private static String mostPopularMovieReviewedByKUsers(IMoviesStorage storage,
                                    int numOfUsers) {
        try {
            return storage.mostPopularMovieReviewedByKUsers(numOfUsers);
        } catch (Exception e) {
            return "";
        }
    }
}
```

1.3 Sample input

You provided with 3 sample input files for the check.

1. **movies-simple1.txt**
2. **movies-simple2.txt**
3. **movies-simple3.txt**

1.4 Output for comparison

Since each input file will produce 4 different outputs you are provided with 12 outputs, so you will be able to compare your and expected output.

1. For **movies-simple1.txt**:
   (a) **queries_out1_1.txt**
   (b) **queries_out1_2.txt**
   (c) **queries_out1_3.txt**

(d) **queries_out1_4.txt**

2. For **movies-simple2.txt**:
   (a) **queries_out2_1.txt**
   (b) **queries_out2_2.txt**
   (c) **queries_out2_3.txt**
   (d) **queries_out2_4.txt**
3. For **movies-simple3.txt**:
   (a) **queries_out3_1.txt**
   (b) **queries_out3_2.txt**
   (c) **queries_out3_3.txt**
   (d) **queries_out3_4.txt**

1.5 How to check yourself

As published in the assignment first of all you need to run following procedure

1. Create a virtual box by:

   ```
   vagrant up
   ```

   It might take a frew minutes for very first time and require connection to the internet. Virtual machine base image will be downloaded and provisioned by vagrant with commands from the **Vagrantfile**.

2. Connect to the virtual box:

   ```
   vagrant ssh
   ```

3. Go to the project folder:

   ```
   cd /homework/vagrant/hw1
   ```

4. Execute maven compilations and build goals:

   ```
   mvn clean install
   ```

   After execution of these command the entire project will be compiled and validated, so you need to wait for **BUILD SUCCESS** message. Once you get the message that means project has been successefully compiled.

5. Execute your implementation on sample input and validate it:

   ```
   mvn exec:java && ./grade.sh
   ```

   You should see the output:

   ```
   Your output is correct. PERFECT MATCH!!!
   ```

   This will indicate correctness of your code otherwise it will present you a differences in the output. This part provides you with (76pts) of the total grade.

Next:

1. Copy pom.xml file into root project folder and add your dependencies whenever needed.
2. Copy **GradingReviewsQueryRunner.java** file into **univ.bigdata.course** package.
3. Copy all input files into **src/main/resources folder**.
4. Once done then repeat steps 1-4, not including 5.
5. In order to produce new output based on new main file run:

```
mvn exec:java \
 -Dexec.mainClass="univ.bigdata.course.GradingReviewsQueryRunner" \
 -Dexec.args="-inputFile=movies-simple${i}.txt -outputFile=queries_out${i}"
```

   where index **i** stends for index of input sample you've been provided with. For example:

```
mvn exec:java \
 -Dexec.mainClass="univ.bigdata.course.GradingReviewsQueryRunner" \
 -Dexec.args="-inputFile=movies-simple1.txt -outputFile=queries_out1"
```

6. Repeat #5 for all inputs
7. You will be results with 12 output files, you need to check each one with corresponding expected sample, each successefull and **EXACT** match gives you 2pts to the final grade, therefore matching all sample inputs gives you 24 additional pts.

Chapter 2

How to submit appeals

1. Appeals accepted in electronic format only, there no frontal appealing session will be held. You need to send an email with clear justification of why and where you think your submission suppose to get an additional grades.
2. Appeals asking to received points while has failed in the tests described above won't be accepted at any case, there is a chance to fix the error and improve the grade, see below.
3. There is an additional chance to increase the grade in case of the failure in the tests above. You need clearly explain the BUG and provide small and self contained fix for the it, while sending the appeal.
4. Due date for the submission is 25/05/2016, no appeals will be accepted afterwards.
5. Please submit appeals in PDF format to the email: **artem.barger@gmail.com**, subject should be **[BigData Course]: Appeal/HW1** (emails with different subjects will be filtered/rejected). Also email has to include all details of the team members.