Faculty of Computers, Informatics and Microelectronics

Technical University of Moldova

APPOO

Laboratory work#2

Using Java Collections in data processing

Author:
Denis POSTICA

Supervisor:
Anastasia SERSUN

May 25, 2017

Laboratory work #2

1 Purpose of the laboratory work

In this work you will need to select a data from a certain domain and analyze the correlation between indicators of this field.

As a result you will need to present what indicators are most correlated in the selected dataset.

Please, select one domain that you would like to investigate and find different indicators in that field. Please, select a filed with statistics for at least 7 different parameters for each entry.

Save data in a file, from which you will load data to your program.

Create program that will:

- load data from the file.
- find correlation coefficient for each pair of indicators of the presented data set.
- present results in readable way.

2 Laboratory work implementation

To show the practical use of Java Collections I created a program that reads a CSV file from a given location, processes this file, creates collection of Java Beans for each row. These beans are processed and processed data is used to calculate correlation. As correlation calculator I used the library from Apache Commons Math (PearsonCorrelation). The code of the program can be found at Git repository `https://github.com/PosticaDenis/APPOO`. The file I used for program I got at `https://www.healthdata.gov/dataset/cms-medicare-and-medicaid-ehr-incentive-program-electronic` It contains some medical data as I understood. The file contains about 50k rows of data so it will take some seconds to read, process all the data.

Laboratory work analysis

To parse the CSV file I used uniVocity-parsers (from `https://github.com/uniVocity/univocity-pa`
The data file is located under resources/MU_REPORT.csv

```java
package APPOO.lab2;

import com.sun.istack.internal.Nullable;
import com.univocity.parsers.common.processor.BeanListProcessor;
import com.univocity.parsers.csv.CsvParser;
import com.univocity.parsers.csv.CsvParserSettings;

import java.io.*;
import java.util.List;

/**
 * Created by Dennis on 17-May-17.
 **/
public class MedicalDataSchema {

    private List<MedicalDataBean> medicalDataBeans;
    private String pathToCSV;

    MedicalDataSchema(String path) {
        pathToCSV = path;
        loadData();
    }

    private void loadData() {

        // BeanListProcessor converts each parsed row to an instance of a given class,
           then stores each instance into a list.
        BeanListProcessor<MedicalDataBean> rowProcessor = new
            BeanListProcessor<>(MedicalDataBean.class);

        CsvParserSettings parserSettings = new CsvParserSettings();
        parserSettings.setRowProcessor(rowProcessor);
        parserSettings.setHeaderExtractionEnabled(true);

        CsvParser parser = new CsvParser(parserSettings);
        parser.parse(getFileReader(pathToCSV));

        // The BeanListProcessor provides a list of objects extracted from the input.
        medicalDataBeans = rowProcessor.getBeans();
        System.out.println("Done loading file.\n");
    }

    @Nullable
    private Reader getFileReader(String absolutePath) {
        try {
            return new InputStreamReader(new FileInputStream(new File(absolutePath)),
                "UTF-8");
        }
        catch (IOException e) {
            System.out.print("Error");
        }
        return null;
    }

    public List<MedicalDataBean> getMedicalDataBeans(){
        return medicalDataBeans;
    }
}
```

```
package APPOO.lab2;


import com.univocity.parsers.annotations.Parsed;

/**
 * Created by Dennis on 17-May-17.
 **/
public class MedicalDataBean {

        @Parsed(index = 0)
```

```
private double npi;
@Parsed(index = 3)
private String bussinesStateTeritory;
@Parsed(index = 4)
private String zip;
@Parsed(index = 5)
private String speciality;
@Parsed(index = 8)
private double programYear;
@Parsed(index = 9)
private String providerStageNumber;
@Parsed(index = 20)
private String productClassification;
@Parsed(index = 21)
private String productSetting;

public double getNpi() {
    return npi;
}

public String getBussinesStateTeritory() {
    return bussinesStateTeritory;
}

public String getZip() {
    return zip;
}

public String getSpeciality() {
    return speciality;
}

public double getProgramYear() {
    return programYear;
}

public String getProviderStageNumber() {
    return providerStageNumber;
}

public String getProductClassification() {
    return productClassification;
}
```

```
        public String getProductSetting() {
            return productSetting;
        }
}
```

The class MedicalDataSchema is responsible for reading and processing the data from file into beans. The uniVocity CVS parser works in the following way: there is a bean (MedicalDataBean in my case) that defines the data from a row of the CSV file; to read just the columns I needed I used @Parsed(index = i), where i is the column from which data will be read and write into the variable declared after; BeanListProcessor converts each parsed row to an instance of a given class, then stores each instance into a list. As result we have a list of beans, where the size of the list is the number of row in the data file.

More as an experiment I use the following data processor to be able to find correlation between String data. I process all the data from the list of Medical Data Beans, for Strings I call a method which returns the id of that String. To get the id I insert all the data into an SortedSet, as we know in a set we can not have duplicate values (this way I filter the data), and later call a method to get the id of each unique String in the SortedSet. These data will be used later for correlation calculation.

```
package APPOO.lab2;

import java.util.*;

/**
 * Created by Dennis on 17-May-17.
 **/
public class DataProcessor {

    private List<MedicalDataBean> medicalDataBeans;
    private List<ProcessedMedicalDataBean> processedMedicalDataBeans = new ArrayList<>();
    SortedSet<String> hashSet = new TreeSet<>();

    DataProcessor(String path) {

        MedicalDataSchema medicalDataSchema = new MedicalDataSchema(path);
        medicalDataBeans = medicalDataSchema.getMedicalDataBeans();
        copyData();
    }

    private void copyData(){
```

```
    populateHashSet();

    for (MedicalDataBean medicalDataBean: medicalDataBeans) {
        ProcessedMedicalDataBean processedMedicalDataBean = new
            ProcessedMedicalDataBean();
        processedMedicalDataBean.setNpi(medicalDataBean.getNpi());
        processedMedicalDataBean.setBussinesStateTeritoryId(getId(medicalDataBean.getBussin
        processedMedicalDataBean.setZip(getId(medicalDataBean.getZip()));
        processedMedicalDataBean.setSpecialityId(getId(medicalDataBean.getSpeciality()));
        processedMedicalDataBean.setProgramYear(medicalDataBean.getProgramYear());
        processedMedicalDataBean.setProviderStageNumberId(getId(medicalDataBean.getProvider
        processedMedicalDataBean.setProductClassificationId(getId(medicalDataBean.getProduc
        processedMedicalDataBean.setProductSettingId(getId(medicalDataBean.getProductSettin

        processedMedicalDataBeans.add(processedMedicalDataBean);
    }
}

private void populateHashSet(){

    for (MedicalDataBean medicalDataBean: medicalDataBeans) {
        setId(medicalDataBean.getZip());
        setId(medicalDataBean.getBussinesStateTeritory());
        setId(medicalDataBean.getSpeciality());
        setId(medicalDataBean.getProviderStageNumber());
        setId(medicalDataBean.getProductClassification());
        setId(medicalDataBean.getProductSetting());
    }
}

private void setId(String data){
    if (data != null && !data.equals("")) {
        hashSet.add(data);
    }
}

private double getId(String data) {

    if (data == null ){
        return -1.0;
    }
    return hashSet.contains(data)? hashSet.headSet(data).size(): -1;
}
```

```
    public List<ProcessedMedicalDataBean> getProcessedMedicalDataBeans() {
        return processedMedicalDataBeans;
    }
}
```

The tool CorrelationCalculator is used to store data from beans into arrays of double and later calculate and output the correlation between column from the processed file.

```
package APPOO.lab2.tools;

import APPOO.lab2.DataProcessor;
import APPOO.lab2.ProcessedMedicalDataBean;
import org.apache.commons.math3.stat.correlation.PearsonsCorrelation;

import java.util.List;

/**
 * Created by Dennis on 20-May-17.
 **/
public class CorrelationCalculator {

    private PearsonsCorrelation correlation = new PearsonsCorrelation();
    private DataProcessor dataProcessor;
    private double[][] columns;

    public CorrelationCalculator(DataProcessor dataProcessor) {
        this.dataProcessor = dataProcessor;
        this.columns = new
            double[8][dataProcessor.getProcessedMedicalDataBeans().size()];
        setColumns();
    }

    private void setColumns() {

        List<ProcessedMedicalDataBean> processedMedicalDataBeans =
            dataProcessor.getProcessedMedicalDataBeans();
        int i = 0;

        for (ProcessedMedicalDataBean data: processedMedicalDataBeans) {
            setValues(data, i);
            i++;
        }
    }
    private void setValues(ProcessedMedicalDataBean data, int i) {
        columns[0][i] = data.getNpi();
        columns[1][i] = data.getBussinesStateTeritoryId();
        columns[2][i] = data.getZip();
        columns[3][i] = data.getSpecialityId();
        columns[4][i] = data.getProgramYear();
        columns[5][i] = data.getProviderStageNumberId();
        columns[6][i] = data.getProductClassificationId();
        columns[7][i] = data.getProductSettingId();

    }

    public void getResults(){

        System.out.println("The results are: ");

        // if the result is NaN it means that the variation of variables in one of the
            column is too small to be taken into consideration.
        for (int i = 0; i<7; i++) {
            System.out.println(correlation.correlation(columns[i], columns[i+1]));
        }
    }
}
```

To get the results we just create a DataProcessor with parameter the path to the file, also a new CorrelationCalculator and give to it the processed Medical Data Beans. Then, we just print the calculated results.

Conclusions

The Java Collections are widely used by any programmer as they provide methods for better, faster data manipulation in a program. In my program I also used Java Collection and this made the work much easier, having all the necessary tools, methods for an effective data manipulation.

References

1 uniVocity documentation, `https://github.com/uniVocity/univocity-parsers`

2 Java - Collections Framework,
`https://www.tutorialspoint.com/java/java_collections.htm`

3 Guide for laboratory work 2.