**Programming Systems and Environments – Lab 4**

Jakub Grzana, 241530

**Task 1 – FileDataGenerator**

Task description includes names bundled with random values assigned, so I’ve decided to reuse Person class from previous laboratories (which later came out advantageous due to People class I’ve created) I’ve expanded this class with methods ‘ToString()’ and ‘FromString()’ which essentially allows to save/load Person in simple string instead of JSON. Then, I’ve created FileDataGenerator that loads two files (“names.txt”, “surnames.txt”, each line contains single name/surname, so it’s possible to make Spanish Esteban Julio Ricardo Montoya de la Rosa Ramirez) and allows to create virtually any number of random People saved in any number of files (one file can contain multiple people).

Then I’ve created FileDataParser that loads all files in given directory and extracts information from them. Internally, it uses People class to calculate average salary or find oldest people in any particular file, or whole database. I’ve added CSV support too.

FileDataGenerator generates sequential filenames, but FileDataParser just loads all files within directory. Filenames doesn’t have to be sequential.

**Task 2 – NumericDataGenerator**

I’ve implemented NumericDataGenerator that generates N random (long) values (atleast 10 digits) and saves them to csv file. Then NumericDataParser can load that CSV file, perform prime check on each of those values, and return list of primes.

Prime check uses slightly more efficient algorithm than typical check of all divisors until sqrt(n), but it isn’t probability based check.

There’s minor bug in my solution: sometimes random number returned by generator is 9 digits long. It should be mathematically impossible, given I’ve used formula: for . I can’t explain why it happens.

**Conclusions**

I don’t know if reading entire CSV file into string, replacing commas with spaces, splitting them and converting one by one into long value is proper way of saving/loading numbers in CSV with Java, but it did work. One thing I notices was: converting float to string used comma, which didn’t work well with CSV files…. So I took the output string and replaced comma with dot. I don’t know what else to say, tasks were easy but time consuming.

**Bibliography:**

Prime check [access 12.04.2022]:

<https://www.geeksforgeeks.org/java-program-to-check-if-a-number-is-prime-or-not/>

Replacing parts of string [access 12.04.2022]:

<https://careerkarma.com/blog/java-string-replace/>

Reading entire file to single string [access 12.04.2022]:

<https://howtodoinjava.com/java/io/java-read-file-to-string-examples/>

Using hashmaps [access 12.04.2022]:

<https://stackoverflow.com/questions/13543457/how-do-you-create-a-dictionary-in-java>

<https://stackoverflow.com/questions/3626752/key-existence-check-in-hashmap>

Get list of files in directory [access 12.04.2022]:

<https://www.baeldung.com/java-list-directory-files>

**CODE: Lab4.java**

import java.io.File;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.nio.file.Files;

import java.nio.file.Paths;

import java.util.ArrayList;

import java.util.Scanner;

public class NumericDataParser {

final private String path;

private ArrayList<Long> numbers = new ArrayList<>();

private static boolean isPrime(long n)

{

// Corner cases

if (n <= 1)

return false;

if (n <= 3)

return true;

// This is checked so that we can skip

// middle five numbers in below loop

if (n % 2 == 0 || n % 3 == 0)

return false;

for (long i = 5; i \* i <= n; i = i + 6)

if (n % i == 0 || n % (i + 2) == 0)

return false;

return true;

}

NumericDataParser(String path)

{

this.path = path;

}

private static String readAllBytesJava7(String filePath)

{

String content = "";

try

{

content = new String ( Files.readAllBytes( Paths.get(filePath) ) );

}

catch (IOException e)

{

e.printStackTrace();

}

return content;

}

public void load()

{

this.numbers.clear();

String content = readAllBytesJava7(this.path);

content = content.replace(",", " ");

String literals[] = content.split(" ");

for(String s : literals)

{

long r = Long.parseLong(s);

this.numbers.add(r);

}

}

public void save() throws Exception

{

String out = "";

for(Long n : this.numbers)

{

out = out + String.format("%d,", n);

}

byte[] bytes = out.getBytes();

Files.write(Paths.get(this.path), bytes);

}

public void selectPrimes()

{

ArrayList<Long> primes = new ArrayList<>();

for(Long n : this.numbers)

{

if(isPrime(n))

primes.add(n);

}

this.numbers = primes;

}

public int size() { return this.numbers.size(); }

@Override public String toString()

{

String output = "";

for(Long n : this.numbers)

{

output = output + String.format("%d\n", n);

}

return output;

}

}

**CODE: FileDataGenerator.java**

import java.io.File;

import java.io.FileNotFoundException;

import java.nio.file.Files;

import java.nio.file.Paths;

import java.util.ArrayList;

import java.util.Scanner;

public class FileDataGenerator {

final private String path;

final private ArrayList<String> names;

final private ArrayList<String> surnames;

public static ArrayList<String> Load(File file) throws Exception // Load file line by line, into array of strings

{

ArrayList<String> output = new ArrayList<>();

try (Scanner myReader = new Scanner(file)) {

while(myReader.hasNextLine())

{

String data = myReader.nextLine();

output.add(data);

}

}

catch(FileNotFoundException e)

{

System.out.println("An error occurred.");

e.printStackTrace();

}

return output;

}

private void ensureDir()

{

File directory = new File(this.path);

if (!directory.exists()){

directory.mkdir();

}

}

public void generate(int fnum, int pnum) throws Exception

{

this.ensureDir();

for(int i = 0; i < fnum; ++i)

{

String filepath = this.path + "/" + i + ".txt";

File newfile = new File(filepath);

newfile.createNewFile();

String out = "";

for(int j = 0; j < pnum; ++j)

{

Person p = new Person(names, surnames, 15, 42, 150, 200);

out = out + p.ToString();

}

byte[] bytes = out.getBytes();

Files.write(Paths.get(filepath), bytes);

}

}

FileDataGenerator(String path) throws Exception

{

this.names = Load(new File("names.txt"));

this.surnames = Load(new File("surnames.txt"));

this.path = path;

}

}

**CODE: FileDataParser.java**

import java.util.\*;

import java.io.File;

import java.io.FileNotFoundException;

import java.nio.file.DirectoryStream;

import java.nio.file.Files;

import java.nio.file.Path;

import java.nio.file.Paths;

import java.util.ArrayList;

import java.util.Scanner;

public class FileDataParser {

private People people = new People();

private Map<String, People> people\_sub = new HashMap<>();

private Set<String> list\_of\_files;

public static ArrayList<String> Load(File file) throws Exception // Load file line by line, into array of strings

{

ArrayList<String> output = new ArrayList<>();

try (Scanner myReader = new Scanner(file)) {

while(myReader.hasNextLine())

{

String data = myReader.nextLine();

output.add(data);

}

}

catch(FileNotFoundException e)

{

System.out.println("An error occurred.");

e.printStackTrace();

}

return output;

}

public Set<String> ListFilesInDirectory(String dir) throws Exception {

Set<String> fileList = new HashSet<>();

try (DirectoryStream<Path> stream = Files.newDirectoryStream(Paths.get(dir))) {

for (Path path : stream) {

if (!Files.isDirectory(path)) {

fileList.add(path.getFileName()

.toString());

}

}

}

return fileList;

}

public int size() { return this.people.size(); }

public String oldest() {

String output = "";

for(String filename : this.list\_of\_files)

{

People local\_ppl = this.people\_sub.get(filename);

local\_ppl.sortByAge();

local\_ppl.reverse();

Person oldest = local\_ppl.getPersonList().get(0);

output = output + String.format("Oldest in %s: %s %s %d\n", filename, oldest.getName(), oldest.getSurname(), oldest.getAge());

}

this.people.sortByAge();

this.people.reverse();

Person oldest = this.people.getPersonList().get(0);

output = output + String.format("Oldest in entire set: %s %s %d", oldest.getName(), oldest.getSurname(), oldest.getAge());

return output;

}

public void oldestCSV(String csv\_filename) throws Exception

{

String output = "";

for(String filename : this.list\_of\_files)

{

People local\_ppl = this.people\_sub.get(filename);

local\_ppl.sortByAge();

local\_ppl.reverse();

Person oldest = local\_ppl.getPersonList().get(0);

output = output + String.format("%s,%s,%s,%d\n", filename, oldest.getName(), oldest.getSurname(), oldest.getAge());

}

this.people.sortByAge();

this.people.reverse();

Person oldest = this.people.getPersonList().get(0);

output = output + String.format("total, %s, %s, %d\n", oldest.getName(), oldest.getSurname(), oldest.getAge());

byte[] bytes = output.getBytes();

Files.write(Paths.get(csv\_filename), bytes);

}

public String averageSalary() {

String output = "";

for(String filename : this.list\_of\_files)

{

People local\_ppl = this.people\_sub.get(filename);

double average = local\_ppl.averageSalary();

String av = String.format("%f", average).replace(",", ".");

output = output + String.format("Average salary in %s: %s\n", filename, av);

}

double average = this.people.averageSalary();

output = output + String.format("Average salary in total: %f\n", average);

return output;

}

public void averageSalaryCSV(String csv\_filename) throws Exception

{

String output = "";

for(String filename : this.list\_of\_files)

{

People local\_ppl = this.people\_sub.get(filename);

double average = local\_ppl.averageSalary();

String av = String.format("%f", average).replace(",", ".");

output = output + String.format("%s,%s\n", filename, av);

}

double average = this.people.averageSalary();

output = output + String.format("total, %f", average);

byte[] bytes = output.getBytes();

Files.write(Paths.get(csv\_filename), bytes);

}

FileDataParser(String path) throws Exception

{

this.list\_of\_files = ListFilesInDirectory(path);

for(String filename : this.list\_of\_files)

{

ArrayList<String> lines = Load(new File(path + "/" + filename));

for(String line : lines)

{

Person p = Person.FromString(line);

this.people.addPerson(p);

if(this.people\_sub.get(filename) == null) { this.people\_sub.put(filename, new People()); }

this.people\_sub.get(filename).addPerson(p);

}

}

}

}

**CODE: NumericDataGenerator.java**

import java.nio.file.Files;

import java.nio.file.Paths;

import java.util.ArrayList;

import java.security.SecureRandom;

public class NumericDataGenerator {

final private String path;

private ArrayList<Long> numbers = new ArrayList<>();

private static SecureRandom randomGenerator = new SecureRandom();

private static long GetRandomNum(long min, long max)

{

final long rand\_val = randomGenerator.nextLong();

return rand\_val % (max - min + 1) + min;

}

public void generate(int num)

{

numbers.clear();

for(int i = 0; i < num; ++i)

{

long r = GetRandomNum(1000000000L, 9999999999L);

this.numbers.add(r);

}

}

public void save() throws Exception

{

String out = "";

for(Long n : this.numbers)

{

out = out + String.format("%d,", n);

}

byte[] bytes = out.getBytes();

Files.write(Paths.get(this.path), bytes);

}

NumericDataGenerator(String path)

{

this.path = path;

}

}

**CODE: NumericDataParser.java**

import java.io.File;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.nio.file.Files;

import java.nio.file.Paths;

import java.util.ArrayList;

import java.util.Scanner;

public class NumericDataParser {

final private String path;

private ArrayList<Long> numbers = new ArrayList<>();

private static boolean isPrime(long n)

{

// Corner cases

if (n <= 1)

return false;

if (n <= 3)

return true;

// This is checked so that we can skip

// middle five numbers in below loop

if (n % 2 == 0 || n % 3 == 0)

return false;

for (long i = 5; i \* i <= n; i = i + 6)

if (n % i == 0 || n % (i + 2) == 0)

return false;

return true;

}

NumericDataParser(String path)

{

this.path = path;

}

private static String readAllBytesJava7(String filePath)

{

String content = "";

try

{

content = new String ( Files.readAllBytes( Paths.get(filePath) ) );

}

catch (IOException e)

{

e.printStackTrace();

}

return content;

}

public void load()

{

this.numbers.clear();

String content = readAllBytesJava7(this.path);

content = content.replace(",", " ");

String literals[] = content.split(" ");

for(String s : literals)

{

long r = Long.parseLong(s);

this.numbers.add(r);

}

}

public void save() throws Exception

{

String out = "";

for(Long n : this.numbers)

{

out = out + String.format("%d,", n);

}

byte[] bytes = out.getBytes();

Files.write(Paths.get(this.path), bytes);

}

public void selectPrimes()

{

ArrayList<Long> primes = new ArrayList<>();

for(Long n : this.numbers)

{

if(isPrime(n))

primes.add(n);

}

this.numbers = primes;

}

public int size() { return this.numbers.size(); }

@Override public String toString()

{

String output = "";

for(Long n : this.numbers)

{

output = output + String.format("%d\n", n);

}

return output;

}

}

**CODE: Person.java**

import java.lang.\*;

import java.util.ArrayList;

import java.security.SecureRandom;

public class Person {

private int age;

private int salary;

private String name;

private String surname;

private static SecureRandom randomGenerator = new SecureRandom();

private void setAge(int age) { this.age = age; }

public int getAge() { return this.age; }

private void setName(String name) { this.name = name; }

public String getName() { return this.name; }

private void setSurname(String surname) { this.surname = surname; }

public String getSurname() { return this.surname; }

private void setSalary(int salary) { this.salary = salary; }

public int getSalary() { return this.salary; }

private static String GetRandomName(ArrayList<String> names)

{

final int index = randomGenerator.nextInt(names.size());

return names.get(index);

}

private static String GetRandomSurname(ArrayList<String> names)

{

final int index = randomGenerator.nextInt(names.size());

return names.get(index);

}

private static int GetRandomNum(int min, int max)

{

final int diff = Math.abs(max - min);

final int rand\_val = randomGenerator.nextInt(diff);

return min + rand\_val;

}

Person()

{

ArrayList<String> names = new ArrayList<>();

names.add("Jagoda");

names.add("Eliza");

names.add("Harold");

names.add("Levi");

names.add("Samuel");

names.add("Igor");

names.add("Ewelina");

names.add("MUNDO");

ArrayList<String> surnames = new ArrayList<>();

surnames.add("Spirit");

surnames.add("Orzeszkowa");

surnames.add("Ackermann");

surnames.add("Ortega");

surnames.add("Light");

surnames.add("Zawicki");

surnames.add("Nerina");

this.setName(GetRandomName(names));

this.setSurname(GetRandomSurname(surnames));

this.setAge(GetRandomNum(18,24));

this.setSalary(GetRandomNum(10, 10000));

}

Person(ArrayList<String> names, ArrayList<String> surnames, int minAge, int maxAge, int minSalary, int maxSalary)

{

this.setName(GetRandomName(names));

this.setSurname(GetRandomSurname(surnames));

this.setAge(GetRandomNum(minAge,maxAge));

this.setSalary(GetRandomNum(minSalary, maxSalary));

}

Person(int age, String name, String surname, int salary)

{

this.setAge(age);

this.setName(name);

this.setSurname(surname);

this.setSalary(salary);

}

@Override public String toString()

{

return String.format("======\nName: %s\nSurname: %s\nAge: %d\nSalary: %d", this.getName(), this.getSurname(), this.getAge(), this.getSalary());

}

public String ToString()

{

return String.format("%s %s %d %d\n", this.getName(), this.getSurname(), this.getAge(), this.getSalary());

}

public static Person FromString(String text) throws Exception

{

String output[] = text.split(" ");

String name = (String) output[0];

String surname = (String) output[1];

int age = Integer.parseInt(output[2]);

int salary = Integer.parseInt(output[3]);

return new Person(age, name, surname, salary);

}

}

**CODE: People.java**

import java.lang.Math;

import java.util.function.Consumer;

import java.util.\*;

public class People {

private List<Person> personList = new ArrayList<>();

public List<Person> getPersonList() { return this.personList; }

public void addPerson(Person p) { this.personList.add(p); }

public int size() { return this.personList.size(); }

People() {}

People(int n)

{

for(int i = 0; i < n; ++i)

{

this.addPerson(new Person());

}

}

public void sort(Comparator<Person> cmp)

{

Collections.sort(this.personList, cmp);

}

public void sortByAge()

{

this.sort( (p1, p2) -> p1.getAge() - p2.getAge() );

}

public void sortBySalary()

{

this.sort( (p1, p2) -> p1.getSalary()- p2.getSalary() );

}

public void sortByName()

{

this.sort( (p1, p2) -> p1.getName().compareTo(p2.getName()) );

}

public void sortBySurname()

{

this.sort( (p1, p2) -> p1.getSurname().compareTo(p2.getSurname()) );

}

public void sortByNameAndSurname()

{

this.sort( (p1, p2) -> {

String n1 = String.format("%s %s", p1.getName(), p1.getSurname());

String n2 = String.format("%s %s", p2.getName(), p2.getSurname());

return n1.compareTo(n2);

});

}

public void reverse()

{

Collections.reverse(this.getPersonList());

}

public People selectFirst(int num)

{

int mx = Math.min(num, this.getPersonList().size());

People ppl = new People();

for(int i = 0; i < mx; ++i)

{

ppl.addPerson(this.getPersonList().get(i));

}

return ppl;

}

public People selectSalary(int salary)

{

People ppl = new People();

this.sortBySalary();

this.reverse();

for(Person p : this.getPersonList())

{

if(p.getSalary() > salary)

{

ppl.addPerson(p);

}

}

return ppl;

}

public double averageSalary()

{

double output = 0;

for(Person p : this.getPersonList())

{

output = output + p.getSalary();

}

return output / this.getPersonList().size();

}

@Override public String toString()

{

String output = "";

for(Person p : this.getPersonList())

{

output = output + p + "\n";

}

return output;

}

}