|  |
| --- |
|  |
| Assignment 1 |
| CSCI 291T - 01 Bioinformatics Computing |

|  |
| --- |
| *Mohamad Mahdi Ziaee - 109561770*  2-4-2016 |

Contents

[/Assignment1/src/com/bio/main/MainProcessor.java 2](#_Toc442345584)

[/Assignment1/src/com/bio/main/PerformanceMonitor.java 3](#_Toc442345585)

[/Assignment1/src/com/bio/main/Process.java 4](#_Toc442345586)

[/Assignment1/src/com/bio/main/io/FileProcessor.java 9](#_Toc442345587)

[/Assignment1/src/com/bio/main/pojo/Gene.java 12](#_Toc442345588)

[/Assignment1/src/com/bio/main/pojo/RefSeq.java 13](#_Toc442345589)

[/Assignment1/src/com/bio/main/pojo/Strand.java 15](#_Toc442345590)

## /Assignment1/src/com/bio/main/MainProcessor.java

package com.bio.main;

/\*\*

\* Main class to run the program.

\*

\* @author Mohamad Mahdi Ziaee

\*

\*/

public class MainProcessor {

private static final String EXON\_ANNOT\_FILE\_PATH = Process.IO\_PATH + "HG19-refseq-exon-annot-chr1-2016";

private static final String GENE\_ANNOT\_FILE\_PATH = Process.IO\_PATH + "HG19-refseq-gene-annot-filtered";

private static final String CHR1\_FILE\_PATH = Process.IO\_PATH + "chr1.fa";

public static void main(String[] args) {

System.out.println("Starting the process...");

PerformanceMonitor mainPm = new PerformanceMonitor();

Process.getInstance().run(GENE\_ANNOT\_FILE\_PATH, EXON\_ANNOT\_FILE\_PATH, CHR1\_FILE\_PATH);

mainPm.end();

System.out.println("Process ended in " + mainPm);

}

}

## /Assignment1/src/com/bio/main/PerformanceMonitor.java

package com.bio.main;

/\*\*

\* The class can be used to measure the performance of a process.

\*

\* @author Mohamad Mahdi Ziaee

\*

\*/

public class PerformanceMonitor {

private final long startTimestamp;

private long endTimestamp;

/\*\*

\* Use the default constructor just before the process starts.

\*/

public PerformanceMonitor() {

startTimestamp = System.currentTimeMillis();

}

/\*\*

\* Invoke this method at the end of the process.

\*/

public void end() {

endTimestamp = System.currentTimeMillis();

}

/\*\*

\* The method prints the time elapsed in milliseconds

\*

\* @see java.lang.Object#toString()

\*/

@Override

public String toString() {

return String.format("%d milliseconds", endTimestamp - startTimestamp);

}

}

## /Assignment1/src/com/bio/main/Process.java

package com.bio.main;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.io.PrintWriter;

import java.util.ArrayList;

import java.util.List;

import com.bio.main.io.FileProcessor;

import com.bio.main.pojo.Gene;

import com.bio.main.pojo.RefSeq;

import com.bio.main.pojo.Strand;

/\*\*

\* The main class in charge of the whole process.

\*

\* @author Mohamad Mahdi Ziaee

\*

\*/

public class Process {

private static final String SEPARATOR\_DOT = ".";

private static final char INTRON\_N = 'N';

public static final String IO\_PATH = "../Assignment1/io/";

private static Process instance;

private Process() {

}

public static Process getInstance() {

if (instance == null) {

instance = new Process();

}

return instance;

}

/\*\*

\* The main method to be invoked to print the final result.

\*

\* @param geneAnnotFilePath

\* @param exonAnnotFilePath

\* @param chrFilePath

\*/

public void run(String geneAnnotFilePath, String exonAnnotFilePath, String chrFilePath) {

try {

System.out.println("Loading file [" + geneAnnotFilePath + "] to read all the gene annotations");

List<RefSeq> geneAnnots = FileProcessor.getInstance().readAnnorationFile(geneAnnotFilePath, null);

// The map contains the gene annotation as key and all the Exon

// annotations as value

List<Gene> genes = new ArrayList<>();

// We loop through each of the gene annotations.

for (RefSeq geneAnnot : geneAnnots) {

Gene gene = new Gene();

gene.setGeneAnn(geneAnnot);

// Reading Exon annotations

System.out.println("Loading file [" + exonAnnotFilePath + "] for geneID [" + geneAnnot.getId() + "]");

gene.setExonAnns(FileProcessor.getInstance().readAnnorationFile(exonAnnotFilePath, geneAnnot.getId()));

// Extracting gene string

gene.setStr(extractGene(geneAnnot, chrFilePath));

// Replacing the Introns with N

replaceIntronsWithN(gene);

// reverse-complemented the negative strands

reverseSequence(gene);

genes.add(gene);

}

printGenes(genes);

} catch (IOException e) {

System.out.println("Cannot read the file:" + e.getMessage());

}

}

/\*\*

\* For a given gene, the method checks its annotation and if the strand is

\* negative, it will apply reverse-complemented.

\*

\* @param gene

\*/

public void reverseSequence(Gene gene) {

// reversing the sequence if the gene's strand is negative

if (Strand.NEGATIVE.equals(gene.getGeneAnn().getStrand())) {

System.out.println("Reverse-complement process for geneID [" + gene.getGeneAnn().getId() + "] starts ...");

PerformanceMonitor performanceMonitor = new PerformanceMonitor();

StringBuilder stringBuilder = new StringBuilder(gene.getStr());

for (int index = 0; index < gene.getStr().length(); index++) {

stringBuilder.setCharAt(index, swapChar(stringBuilder.charAt(index)));

}

gene.setStr(stringBuilder.reverse().toString());

performanceMonitor.end();

System.out.println("Reverse-complement process ended in " + performanceMonitor);

}

}

/\*\*

\* Swaps characters in a sequence, with respect to the letter cases. It will

\* ignore N character. If unknown character is passed, it will throw a run

\* time exception.

\*

\* @param chr

\* @return

\*/

private char swapChar(char chr) {

if (chr == INTRON\_N) {

return INTRON\_N;

}

if (Character.toLowerCase(chr) == 't') {

return Character.isLowerCase(chr) ? 'a' : 'A';

}

if (Character.toLowerCase(chr) == 'c') {

return Character.isLowerCase(chr) ? 'g' : 'G';

}

if (Character.toLowerCase(chr) == 'a') {

return Character.isLowerCase(chr) ? 't' : 'T';

}

if (Character.toLowerCase(chr) == 'g') {

return Character.isLowerCase(chr) ? 'c' : 'C';

}

throw new RuntimeException("Found an unknown character [" + chr + "]");

}

/\*\*

\* Prints the genes given into the following path: {@link Process#IO\_PATH} +

\* Result.fa with the following format: >chr1.start.end.gene\_ID.strand

\*

\* @param genes

\* @throws FileNotFoundException

\*/

private void printGenes(List<Gene> genes) throws FileNotFoundException {

String fileName = IO\_PATH + "Result.fa";

System.out.println("Exporting the final result to file [" + fileName + "]");

try (PrintWriter out = new PrintWriter(fileName)) {

for (Gene gene : genes) {

String header = ">" + gene.getGeneAnn().getChromosome() + SEPARATOR\_DOT + gene.getGeneAnn().getStart()

+ SEPARATOR\_DOT + gene.getGeneAnn().getEnd() + SEPARATOR\_DOT + gene.getGeneAnn().getId()

+ SEPARATOR\_DOT + gene.getGeneAnn().getStrand().getValue();

System.out.println("Printing the header: [" + header + "]");

out.println(header);

out.println(gene.getStr());

}

}

}

/\*\*

\* Replaces Introns by looking at the Exon annotations given. If the gene

\* sequence is not within the Exon ranges defined, it will be replaced by N

\* (as it is an Intron)

\*

\* @param gene

\*/

public void replaceIntronsWithN(Gene gene) {

System.out.println("Relacing Intros with N for geneID [" + gene.getGeneAnn().getId() + "] starts ...");

PerformanceMonitor pm = new PerformanceMonitor();

char[] charArray = gene.getStr().toCharArray();

for (int index = 0; index < gene.getStr().length(); index++) {

if (!isIndexWithinExon(gene, index)) {

// Replacing N if we are outside the Exon index (if it is

// Intron)

charArray[index] = INTRON\_N;

}

}

gene.setStr(new String(charArray));

pm.end();

System.out.println("Replacing Introns with N was done in " + pm);

}

/\*\*

\* Returns true if the index given is within any of the Exon genes ranges in

\* annotation file.

\*

\* @param gene

\* @param index

\* @return

\*/

private boolean isIndexWithinExon(Gene gene, int index) {

// We add the starting point of the gene to the current index to find

// the real index.

int actualIndex = index + gene.getGeneAnn().getStart();

for (RefSeq refSeq : gene.getExonAnns()) {

if (refSeq.getStart() <= actualIndex && refSeq.getEnd() > actualIndex) {

return true;

}

}

return false;

}

/\*\*

\* Reads Gene string using the RefSeq (gene annotation) id.

\*

\* @param refSeq

\* @param chrFilePath

\* @return

\* @throws IOException

\* @throws FileNotFoundException

\*/

private String extractGene(RefSeq refSeq, String chrFilePath) throws IOException, FileNotFoundException {

System.out.println("Loading file [" + chrFilePath + "] to extract [" + refSeq.getId() + "] gene ...");

PerformanceMonitor extractGenePm = new PerformanceMonitor();

String gene = FileProcessor.getInstance().readChromosomeFile(chrFilePath, refSeq);

extractGenePm.end();

System.out.println("Gene extraction was done in " + extractGenePm);

return gene;

}

}

## /Assignment1/src/com/bio/main/io/FileProcessor.java

package com.bio.main.io;

import java.io.BufferedReader;

import java.io.FileInputStream;

import java.io.IOException;

import java.io.InputStreamReader;

import java.nio.file.Files;

import java.nio.file.Paths;

import java.util.ArrayList;

import java.util.List;

import org.apache.commons.io.input.BoundedInputStream;

import com.bio.main.pojo.RefSeq;

/\*\*

\* The main class in charge of reading from and writing to a file.

\*

\* @author Mohamad Mahdi Ziaee

\*

\*/

public class FileProcessor {

private static final int CHR1\_FA\_STARTING\_BYTES = 6;

private static final int CHR1\_FA\_COLUMNS = 50;

private static FileProcessor instance;

private FileProcessor() {

}

public static FileProcessor getInstance() {

if (instance == null) {

instance = new FileProcessor();

}

return instance;

}

/\*\*

\* For a given Gene ID, a list of RefSeq will be returned by looking at the

\* line and check if the geneID exists anywhere. If the geneID is not given,

\* the method will read the whole file.

\*

\* @param filePath

\* @param geneId

\* @return

\* @throws IOException

\*/

public List<RefSeq> readAnnorationFile(String filePath, String geneId) throws IOException {

List<RefSeq> result = new ArrayList<>();

for (String line : Files.readAllLines(Paths.get(filePath))) {

// Only add those which we are interested in by filtering using the

// geneId given. If the geneIn is null, we include all.

if (geneId == null || (geneId != null && line.contains(geneId))) {

result.add(new RefSeq(line));

}

}

return result;

}

/\*\*

\* Reads chromosome file based on the RefReq starting and ending indexes.

\*

\* @param filePath

\* @param refSeq

\* @return

\* @throws IOException

\*/

public String readChromosomeFile(String filePath, RefSeq refSeq) throws IOException {

String result = ""; // Final result will be stored in this string.

int start = refSeq.getStart();

int end = refSeq.getEnd();

int calculatedStartIndex = CHR1\_FA\_STARTING\_BYTES + start + getNumberOfNewLines(start) + 1; // Excluding the first index by adding 1

int length = end - start;

FileInputStream file = new FileInputStream(filePath);

// Since the file given is big, that is why we want to skip the

// unnecessary part of the file instead of reading the whole file.

file.skip(calculatedStartIndex);

try (BufferedReader br = new BufferedReader(new InputStreamReader(new BoundedInputStream(file)))) {

int charCount = 0;

int fileChar;

StringBuilder response = new StringBuilder();

while ((fileChar = br.read()) != -1) {

// We want to stop the loop if we have read enough

if (charCount >= length) {

break;

}

// Making sure to only put the letters in the string. This is to

// exclude '\n' (New lines)

if (Character.isLetter(fileChar)) {

response.append((char) fileChar);

charCount++;

}

}

result = response.toString();

}

return result;

}

/\*\*

\* Since Chromosome file contains "\n" (new lines) and it counts as one

\* character, this method helps finding the number of new lines for a given

\* index.

\*

\* @param index

\* @return

\*/

private Integer getNumberOfNewLines(int index) {

return index / CHR1\_FA\_COLUMNS;

}

}

## /Assignment1/src/com/bio/main/pojo/Gene.java

package com.bio.main.pojo;

import java.util.List;

/\*\*

\* Gene class contains gene annotation, the Exons annotation related to the gene

\* and the actual gene sequence data.

\*

\* @author Max

\*

\*/

public class Gene {

private RefSeq geneAnn;

private List<RefSeq> exonAnns;

private String str; // The actual data gene sequence

public RefSeq getGeneAnn() {

return geneAnn;

}

public void setGeneAnn(RefSeq geneAnn) {

this.geneAnn = geneAnn;

}

public String getStr() {

return str;

}

public void setStr(String str) {

this.str = str;

}

public List<RefSeq> getExonAnns() {

return exonAnns;

}

public void setExonAnns(List<RefSeq> exonAnns) {

this.exonAnns = exonAnns;

}

}

## /Assignment1/src/com/bio/main/pojo/RefSeq.java

package com.bio.main.pojo;

import java.util.StringTokenizer;

/\*\*

\* A POJO to keep RefSeq structure.

\*

\* @author Mohamad Mahdi Ziaee

\*

\*/

public class RefSeq {

private String chromosome;

private Integer start;

private Integer end;

private String id;

private Strand strand;

/\*\*

\* Populates the fields in the POJO using the string given which is contains

\* the data separated with tab.

\*

\* @param line

\*/

public RefSeq(String line) {

StringTokenizer st = new StringTokenizer(line, "\t");

this.chromosome = st.nextToken();

this.start = Integer.valueOf(st.nextToken());

this.end = Integer.valueOf(st.nextToken());

this.id = st.nextToken();

st.nextToken(); // Skipping one column

this.strand = Strand.value(st.nextToken());

}

public String getChromosome() {

return chromosome;

}

public void setChromosome(String chromosome) {

this.chromosome = chromosome;

}

public Integer getStart() {

return start;

}

public void setStart(Integer start) {

this.start = start;

}

public Integer getEnd() {

return end;

}

public void setEnd(Integer end) {

this.end = end;

}

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public Strand getStrand() {

return strand;

}

public void setStrand(Strand strand) {

this.strand = strand;

}

}

## /Assignment1/src/com/bio/main/pojo/Strand.java

package com.bio.main.pojo;

/\*\*

\* An enumeration to hold the possible values for Strand

\*

\* @author Mohamad Mahdi Ziaee

\*

\*/

public enum Strand {

POSITIVE("+"), NEGATIVE("-");

private String value;

Strand(String value) {

this.value = value;

}

public String getValue() {

return this.value;

}

public static Strand value(String str) {

for (Strand strand : Strand.values()) {

if (strand.value.equals(str)) {

return strand;

}

}

return null;

}

}