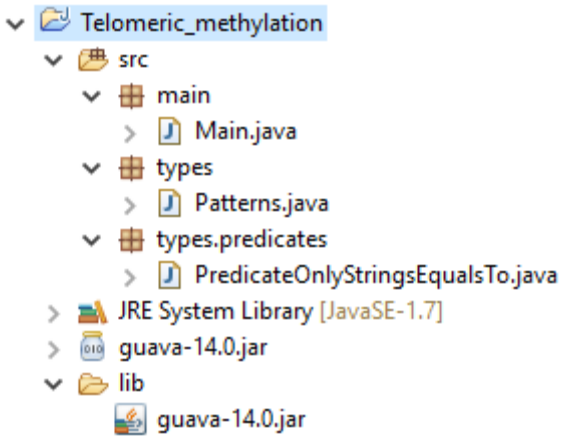
Telomeric analyses source code

This Java application is within a Project entitled “Telomeric methylation”, which is structured in 3 different packages named “main”, “types” and “types.predicates”. There is a class (.java) within each package named “Main.java”, “Patterns.java” and “PredicateOnlyStringsEqualsTo.java”, respectively. The Google “guava” library has been used.
A schematic representation of the project is shown in the following image:



The codes of each class are as follows:

Main.java

```java
package main;

import java.io.IOException;
import java.util.Arrays;
import java.util.List;

import types.Patterns;


public class Main {



	//Name of the file (including the extension)
	private static String file = "SRRXXXXXX(YYYTAAA)4.fasta";
	//Route of the address where the file is found + "file"
	private static String address = "C:/....../" + file ;


	public static void main(String[] args) throws IOException {

		calculate(address);
	}
```

```
	private static void calculate(String dir) throws IOException{
		List<String> AllSequences = Patterns.AllStrings(dir);
		List<String> only50perSequence = Patterns.AllStringsOfOnly(50,
AllSequences);
		List<String> rightSequencesYYYTAAA =
Patterns.RightSecuencesYYYTAAA(only50perSequence);


		Integer YYYTAAA = rightSequencesYYYTAAA.size();
		Integer numberOfC = Patterns.NumberOfC(rightSequencesYYYTAAA);
		Integer noCs =
Patterns.NumberOfCPerSequence(rightSequencesYYYTAAA)[0];

		System.out.println("Run: " + file);

		System.out.println("Number of (YYYTAAA)n reads (denoted as
50TRs)");
		System.out.println(YYYTAAA);

		System.out.println("Number of cytosines whithin 50TRs");
		System.out.println(numberOfC);

		System.out.println("% of cytosines whithin 50TRs");
		Double aux1 = 100*((1.0*numberOfC)/(YYYTAAA*21));
		System.out.println(aux1.floatValue() + "%");

		System.out.println("Number of (TTTTAAA)n reads");
		System.out.println(noCs);

		System.out.println("% of cytosine containing 50TRs");
		Double aux2 = 100-(100*((1.0*noCs)/YYYTAAA));
		System.out.println(aux2.floatValue() + "%");

		System.out.println("Number of 0 to n C containing 50TRs");

	System.out.println(Arrays.toString(Patterns.NumberOfCPerSequence(right
SequencesYYYTAAA)));

		System.out.println("% of 1 to n C containing 50TRs from all the
C containing 50TRs.");

	System.out.println(Arrays.toString(Patterns.PercentageOfCForSequence(P
atterns.NumberOfCPerSequence(rightSequencesYYYTAAA))));

		System.out.println("List of sequences formed by YYYTAAA:");
		System.out.println(rightSequencesYYYTAAA);



	}
}
```

Patterns.java

```
package types;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import com.google.common.collect.Iterables;
import com.google.common.collect.Lists;

import types.predicates.PredicateOnlyStringsEqualsTo;


public class Patterns {

	//Different patterns used below

	private static String first = "((((((A)?A)?(C|T))?(C|T))?(C|T))?T)?";
	private static String middle = "(AAA(C|T)(C|T)(C|T)T)+";
	private static String last =
"(A(A(A(((C|T)((C|T)((C|T)(T)?)?)?)?)?)?)?)?";
	private static Pattern patternYYYTAAA = Pattern.compile("^" + first +
middle + last + "$");



	//Obtention of DNA strings from the file. Lines containing no DNA
strings have one of the following characters
	// '@','>','+' or contain the word 'length'
	//If the line does not contain one of the above mentioned characters
it will be considered a DNA string.

	public static List<String> AllStrings(String nameFile) throws
IOException{
		List<String> aux = new ArrayList<String>();
		BufferedReader br = new BufferedReader(new
FileReader(nameFile));
		String line = br.readLine();
		while(line!=null){
			processLine(line,aux);
			line = br.readLine();
		}
		br.close();
		return aux;
	}
```

```
	private static void processLine(String line, List<String> aux) {
		if((!line.contains("@")) && (!line.contains(">")) &&
(!line.equals("")) && (!line.contains("+")) && (!line.contains("length"))){
			aux.add(line);
		}

	}


	//Archives had different sequence lengths. We used the first sequence
as a reference of the rest.
	//the following method generates a list with 50 bp long sequences from
every file.


	public static List<String> AllStringsOfOnly(Integer
finalNumberOfNucleotids, List<String> list){

		Iterable<String> it = Iterables.filter(list, new
PredicateOnlyStringsEqualsTo(list.get(0).length()));
		List<String> copylist = Lists.newArrayList(it);


		List<String> newlist = new ArrayList<String>();

		Character[] characters = new Character[finalNumberOfNucleotids];

		String aux = "";

		for(String str:copylist){
			for(int i = 0; i<finalNumberOfNucleotids; i++){
				characters[i]= str.charAt(i);
			}
			for(int i = 0; i<finalNumberOfNucleotids; i++){
				aux = aux + characters[i];
			}
			newlist.add(aux);

			aux="";

		}

		return newlist;
	}



	//Obtention of the 50 bp long (YYYTAAA) sequences (50TRs)
	public static List<String> RightSecuencesYYYTAAA(List<String> list) {
		List<String> aux = new ArrayList<String>();

		for (String s : list) {
			Matcher mat = patternYYYTAAA.matcher(s);
			if (mat.find()) {
				aux.add(s);
			}
		}
```

```
        return aux;
}


//Determination of the number of cytosines

public static Integer NumberOfC(List<String> list){

        Integer numberCs = 0;

        for(String s:list){
                for(int i = 0; i<s.length() ; i++){
                        if(s.charAt(i)=='C'){
                                numberCs++;
                        }
                }
        }

        return numberCs;
}


public static Integer[] NumberOfCPerSequence(List<String> list){


        Integer aux = 0;

        Integer[] res = new Integer[23];

        for(int i=0; i<23 ; i++){
                res[i]=0;
        }


        for(String s:list){
                for(int i = 0; i<s.length() ; i++){
                        if(s.charAt(i)=='C'){
                                aux++;
                        }
                }
                res[aux]++;
                aux=0;
        }

        return res;
}


public static String[] PercentageOfCForSequence(Integer[] in){

        Integer total = 0;
        Integer largo = in.length;

        for(Integer integer:in){
                total = total + integer;
```

```
		}
		total = total-in[0];

		String[] ret = new String[largo-1];

		for(int i = 1; i<=ret.length ; i++){
			Double dou = 100.0*((1.0*in[i])/total);
			Float aux = dou.floatValue();
			ret[i-1] = aux + "%";
		}

		return ret;
	}


}
```

PredicateOnliStringsEqualsTo.java

```
package types.predicates;

import com.google.common.base.Predicate;

public class PredicateOnlyStringsEqualsTo implements Predicate<String> {

	private Integer num;

	public PredicateOnlyStringsEqualsTo(Integer num){
		this.num = num;
	}


	public boolean apply(String str) {

		return str.length()==num;
	}

}
```