19 great!

marking-sheet.txt

```
============================================================
marking cits1001 project 2 for ngj04 at 2017-06-05 17:32
============================================================
2:26pm Thu 1st Jun, 2017 130.95.252.115 validation of 5 submitted files SUCCESSFUL

=== Submission and Compilation /2 ==========================            2

@author Joshua Ng (20163079) Mohamed Yusuf(22273364)

Submitted NgramAnalyser has 00108 lines of code and 00063 non-comment lines
Submitted MarkovModel has 00063 lines of code and 00042 non-comment lines
Submitted ModelMatcher has 00085 lines of code and 00052 non-comment lines
Submitted MatcherController has 00075 lines of code and 00051 non-comment lines
Submitted ProjectTest has 00068 lines of code and 00057 non-comment lines

NgramAnalyser compiled successfully
MarkovModel compiled successfully
ModelMatcher compiled successfully
MatcherController compiled successfully

=== JUnit tests (Correctness) /8 ===========================        8

Running MarkerNgramAnalyserTestBasic
OK (5 tests)
Running MarkerMarkovModelTest
OK (3 tests)
Running MarkerModelMatcherTest
OK (1 test)
Running MarkerMatcherControllerTest
OK (1 test)
Running ProjectTest
OK (6 tests)

Marker tests (6 marks)


Coverage of ProjectTest student tests (2 marks)
                                                                   9

=== Clarity and Design / 10 ================================             1
                          ✓         ×
Code is neatly laid out and indented, consistent bracketing, appropriate comments (2 marks)
                           ✓        ✓                                         2
No System.out in project classes. Good variables names. Removed //TODO, dead code (2 marks)
                                                                       3
Appropriate choices of program constructs and programming patterns
(e.g. for-each loops, if without empty branches, exception handling etc)  (3 marks)
                                                                              3
Reuse of code and Java library use (collections .sort, .size, HashSet, StringBuffer)
applied in, for example, alphabetSize and toString methods (3 marks)


=== Extension /2 ===========================================

Extension.pdf not submitted.
```

compile-errors.txt

feedback_tests.txt

```

Running MarkerNgramAnalyserTestBasic
JUnit version 4.12
.....
Time: 0.01

OK (5 tests)

Running MarkerMarkovModelTest
JUnit version 4.12
...
Time: 0.011

OK (3 tests)

Running MarkerModelMatcherTest
JUnit version 4.12
.
Time: 0.01

OK (1 test)

Running MarkerMatcherControllerTest
JUnit version 4.12
.>Explanation is:
This is the best match because it
has the greatest log likelihood compared to the
others its loglikelihood is : -1.4803338724445219
>(explanation end)

Time: 0.014

OK (1 test)

Running ProjectTest
JUnit version 4.12
......
Time: 0.013

OK (6 tests)
```

NgramAnalyser.java

```java
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Set;

import java.util.HashSet;
import java.util.Arrays;

/**
 * Perform n-gram analysis of a string.
 *
 * Analyses the frequency with which distinct n-grams, of length n,
 * appear in an input string. For the purposes of all analyses of the input
 * string, the final n-1 n-grams appearing in the string should be
 * "filled out" to a length of n characters, by adding
 * a sequence of contiguous characters from the start of the string.
 * e.g. "abbc" includes "bca" and "cab" in its 3-grams
 *
 * @author Joshua Ng (20163079), Mohamed Yusuf (22273364)
 * @version 16/5/2017
 */
public class NgramAnalyser
{
    /** dictionary of all distinct n-grams and their frequencies */
    private HashMap<String,Integer> ngram;

    /** number of distinct characters in the input */
    private int alphabetSize;

    /** n-gram size for this object (new field) */
    private int ngramSize;

    /** number of ngram counts (not requiring them to be distinct) */
    private int ngramCount;

    /**
     * Analyse the frequency with which distinct n-grams, of length n,
     * appear in an input string.
     * n-grams at the end of the string wrap to the front
     * e.g. "abbbbc" includes "bca" and "cab" in its 3-grams
     * @param int n size of n-grams to create
     * @param String inp input string to be modelled
     * @throws IllegalArgumentException if the input fields are unsuitable.
     */
    public NgramAnalyser(int n, String inp)
    {
        if(inp == null){throw new IllegalArgumentException(
            "Error : input string cannot be null");}
        if(inp.isEmpty()){ throw new IllegalArgumentException(
            "Error : input string cannot be empty");}
        if(n <= 0){ throw new IllegalArgumentException(
            "Error : ngram size cannot be zero or less than zero");}
        if(n > inp.length()){throw new IllegalArgumentException(
            "Error: your ngram cannot be larger than your input string");}

        ngram = new HashMap<>();
        ngramSize = n;

        char[] inp_array = inp.toCharArray();
        ngramCount = inp_array.length;

        for(int i=0; i < ngramCount; i++) {
            String gram = createGram(inp_array, i, n);
            int freq = getNgramFrequency(gram) + 1;
            ngram.put(gram, freq);
        }

        HashSet<Character> alphabet = new HashSet<>();
        for(char c : inp_array) {
            alphabet.add(c);
        }
        alphabetSize = alphabet.size();

    }

    /**
     * Analyses the input text for n-grams of size 1.
     */
    public NgramAnalyser(String inp)
    {
        this(1,inp);
    }

    /**
     * Creates n-gram string from a char array at p index.
     * n-grams at the end of the string wrap to the front.
     * e.g. createGram({'a', 'b', 'c'}, 0, 2) returns 'ab'
     * @param char[] inp input string array to be modelled
     * @param int p index position of n-gram
     * @param int n size of gram.
     */
    private String createGram(char[] inp, int p, int n)
    {
        char[] gram = new char[n];
        for(int i=0; i<n; i++) {
            gram[i] = inp[(p+i)%inp.length];
        }
        return new String(gram);
    }

    /**
     * @return int the size of the alphabet of a given input
     */
    public int getAlphabetSize() {
        return alphabetSize;
    }

    /**
     * @return the total number of distinct n-grams appearing
     *         in the input text.
     */
    public int getDistinctNgramCount() {
        return ngram.size();
    }

    /**
     * @return Return a set containing all the distinct n-grams
     *         in the input string.
     */
    public Set<String> getDistinctNgrams() {
        return ngram.keySet();
```

✓ Good!

```
    }

    /**
     * @return the total number of n-grams appearing
     *         in the input text (not requiring them to be distinct)
     */
    public int getNgramCount() {
        return ngramCount;
    }

    /** Return the frequency with which a particular n-gram appears
     * in the text. If it does not appear at all, return 0.
     *
     * @param ngram The n-gram to get the frequency of
     * @return The frequency with which the n-gram appears.
     */
    public int getNgramFrequency(String ngram) {
        if(!this.ngram.containsKey(ngram)) {
            return 0;
        }
        else {
            return this.ngram.get(ngram);
        }
    }

    /**
     * Sorts an unorder set to and return a ordered String array.
     *
     * @param un The Set of unordered strings.
     * @return An ordered string array.
     */
    private String[] sortSet(Set<String> un){
        Set<String> unOrder = un;
        String[] order = unOrder.toArray(new String[unOrder.size()]);
        Arrays.sort(order);
        return order;
    }

    /**
     * Generate a summary of the ngrams for this object.
     * @return a string representation of the n-grams in the input text
     * comprising the ngram size and then each ngram and its frequency
     * where ngrams are presented in alphabetical order.
     */
    public String toString()
    {
        String summary = "" + ngramSize;
        String[] order = sortSet(getDistinctNgrams());
        for(String m : order){
            summary += "\n" + m + " " + ngram.get(m);
        }
        summary += "\n";
        return summary;
    }

}
```

MarkovModel.java

```
import java.util.Set;
/**
 * Construct a Markov model of order /k/ based on an input string.
 *
 * @author Joshua Ng (20163079) Mohamed Yusuf(22273364)
 * @version 20/5/2017
 */
public class MarkovModel
{

    /** Markov model order parameter */
    int k;
    /** Input String */
    String s;
    /** ngram model of order k */
    NgramAnalyser ngram;
    /** ngram model of order k+1 */
    NgramAnalyser n1gram;

    /**
     * Construct an order-k Markov model from string s
     * @param k int order of the Markov model
     * @param s String input to be modelled
     * @throws IllegalArgumentException if the input fields are unsuitable.
     */
    public MarkovModel(int k, String s)
    {                          pls use consistent brackets
        if(s == null){throw new IllegalArgumentException(
            "Error : input string cannot be null");}
        if(s.isEmpty()){ throw new IllegalArgumentException(
            "Error : input string cannot be empty");}
        if(k <= 0){ throw new IllegalArgumentException(
            "Error : ngram size cannot be zero or less than zero");}
        if(k+1 > s.length()){throw new IllegalArgumentException(
            "Error: your ngram cannot be larger than your input string");}
        this.s = s;
        this.k = k;
        ngram = new NgramAnalyser(k, s);
        n1gram = new NgramAnalyser(k+1, s);
    }

    /**
     * @return order of this Markov model
     */
    public int getK()
    {
        return k;
    }

    /** Estimate the probability of a sequence appearing in the text
     * using simple estimate of freq seq / frequency front(seq).
     * @param sequence String of length k+1
     * @return double probability of the last letter occuring in the
     * context of the first ones or 0 if front(seq) does not occur.
     */
    public double simpleEstimate(String sequence) {
        if(sequence == null){return (double) 0;}
        if(sequence.isEmpty()){return (double) 0;}

        String subSequence = sequence.substring(0,sequence.length()-1);
        int aa = ngram.getNgramFrequency(subSequence);
        if(aa == 0) {return (double) 0;}
        int aab = n1gram.getNgramFrequency(sequence);
        return (double) aab/aa;
    }
    /**
     * Calculate the Laplacian probability of string obs given this Markov model
     * @para sequence String of length k+1
     * @return double laplace probability of sequence string
     */
    public double laplaceEstimate(String sequence)
    {
        if(sequence == null){return (double) 0;}
        if(sequence.isEmpty()){return (double) 0;}

        String subSequence = sequence.substring(0,sequence.length()-1);
        int aa = ngram.getNgramFrequency(subSequence);
        if((aa+ngram.getAlphabetSize()) == 0) {return (double) 0;}
        int aab = n1gram.getNgramFrequency(sequence);
        return (double) (aab+1)/(aa+ngram.getAlphabetSize());
    }

    /**
     * @return String representing this Markov model
     */
    public String toString()
    {
        String s = "" + k;
        s += "\n" + ngram.getAlphabetSize();
        s += "\n" + ngram.toString() + n1gram.toString();

        return s;
    }

}
```

StringBuilder would be good here, but this is fine

ModelMatcher.java

```java
import java.util.HashMap;
import java.util.Collection;
import java.util.ArrayList;
import java.util.Arrays;
import java.lang.Math;
import java.util.Collections;

/**
 * Report the average log likelihood of a test String occuring in a
 * given Markov model and detail the calculated values behind this statistic.
 *
 * @author Joshua Ng (20163079), Mohamed Yusuf (22273364)
 * @version (20/5/17)
 */
public class ModelMatcher
{

    /** log likelihoods for a teststring under a given model */
    private HashMap<String,Double> logLikelihoodMap;
    /** summary statistic for this setting */
    private double averageLogLikelihood;
    /** given markov model */
    private MarkovModel model;
    /** given input string */
    private String input;

    /**
     * Constructor to initialise the fields for the log likelihood map for
     * a test string and a given Markov model and
     * the average log likelihood summary statistic
     * @param MarkovModel model a given Markov model object
     * @param String teststring
     * @throws IllegalArgumentException if the input fields are unsuitable.
     */
    public ModelMatcher(MarkovModel model, String testString)
    {
        if(testString == null){throw new IllegalArgumentException("Error : input string cannot be null");}
        if(testString.isEmpty()){throw new IllegalArgumentException("Error : input string cannot be empty");}
        if(model == null){throw new IllegalArgumentException("Error : model cannot be null");}
        this.model = model;
        input = testString;
        logLikelihoodMap = new HashMap<>();
        NgramAnalyser testA = new NgramAnalyser(model.getK()+1,testString);
        for(String i : testA.getDistinctNgrams()){
            double estimate = Math.log10(model.laplaceEstimate(i));
            estimate *= testA.getNgramFrequency(i);
            logLikelihoodMap.put(i,estimate);
        }

        averageLogLikelihood = averageLogLikelihood(logLikelihoodMap,
            testA.getNgramCount());
    }

    /** Helper method that calculates the average log likelihood statistic
     * given a HashMap of strings and their Laplace probabilities
     * and the total number of ngrams in the model.
     *
     * @param logs map of ngram strings and their log likelihood
     * @param ngramCount int number of ngrams in the original test string
     * @return average log likelihood: the total of loglikelihoods
     *    divided by the ngramCount
     */
    private double averageLogLikelihood(HashMap<String,Double> logs, int ngramCount)
    {
        return ((totalLogLikelihood(logs))/ngramCount);
    }

    /** Helper method to calculate the total log likelihood statistic
     * given a HashMap of strings and their Laplace probabilities
     * and the total number of ngrams in the model.
     *
     * @param logs map of ngram strings and their log likelihood
     * @return total log likelihood: the sum of loglikelihoods in logs
     */
    private double totalLogLikelihood(HashMap<String,Double> logs)
    {
        double sum = 0;
        if(logs == null){return sum;}

        for(double i : logs.values()){
            sum += i;
        }
        return sum;
    }

    /**
     * @return the average log likelihood statistic
     */
    public double getAverageLogLikelihood()
    {
        return averageLogLikelihood;
    }

    /**
     * Returns the log likelihood value for a given ngram string.
     * @param ngram A string for the ngram
     * @return double probability value for the given ngram.
     */
    public double getLogLikelihood(String ngram)
    {
        if(ngram == null){return 0.0;}
        if(ngram.isEmpty()){ return 0.0;}
        if(!logLikelihoodMap.containsKey(ngram)) {
            return 0.0;
        }
        else {
            return logLikelihoodMap.get(ngram);
        }
    }

    /**
     * Make a String summarising the log likelihood map and its statistics
     * @return String of ngrams and their loglikeihood differences between the models
     * The likelihood table is ordered by ngrams alphabetically
     */
    public String toString()
    {
```

```java
        String s = "averageLogLikelihood of " + input + "\n";
        s += "Context\tCharacter\tlog10 of laplace estimate\n";
        ArrayList<String> ngrams = new ArrayList<>(logLikelihoodMap.keySet());
        Collections.sort(ngrams);
        for(String i : ngrams) {
            s += "\"" + i.substring(0,i.length()-1) + "\"\t'"
            + i.substring(i.length()-1) + "'\t\t" + getLogLikelihood(i) + "\n";
        }
        s += "Average log likelihood " + getAverageLogLikelihood() + "\n";
        return s;
    }

}
```

MatcherController.java

```
import java.io.File;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Set;
import java.io.*;

/** Create and manipulate Markov models and model matchers for lists of training data
 * a test data String and generate output from it for convenient display.
 *
 * @author Mohamed Yusuf (22273364), Joshua Ng (20163079)
 * @version (22/5/17)
 *
 */
public class MatcherController {
    /** list of training data string used to generate markov models */
    ArrayList<String> trainingDataList;
    /** test data to be matched with the models */
    String testData;
    /** order of the markov models*/
    int k;
    /** generated list of markov models for the given training data*/
    ArrayList<MarkovModel> modelList;
    /** generated list of matchers for the given markov models and test data*/
    ArrayList<ModelMatcher> matcherList;

    /** Generate models for analysis
     * @param k order of the markov models to be used
     * @param testData String to check against different models
     * @throw unchecked exceptions if the input order or data inputs are invalid
     */
    public MatcherController(int k, ArrayList<String> trainingDataList, String testData)
    {
        try {
            this.trainingDataList = trainingDataList;
            this.k = k;
            this.testData = testData;
            modelList = new ArrayList<>();
            matcherList = new ArrayList<>();

            for(String i : trainingDataList){
                MarkovModel m = new MarkovModel(k,i);
                modelList.add(m);
            }
            for(MarkovModel i : modelList){
                ModelMatcher m = new ModelMatcher(i,testData);
                matcherList.add(m);
            }
        }

        catch(IllegalArgumentException e){
            displayError("Caught an illegal Argument exception " + e.getMessage());

        }
    }

    /** @return a string containing all lines from a file
     * if file contents can be got, otherwise null
     * This method should process any exceptions that arise.
     */
    private static String getFileContents(String filename) {
        try{
            ArrayList<String> reads = FileIO.readFile(filename);
            String contents = "";
            for(String i : reads){
                contents += i;
            }
            return contents;

        }catch(java.io.FileNotFoundException e){
            return null;
        }
        catch(java.io.IOException e){
            return null;
        }
    }

    /**
     * @return the ModelMatcher object that has the highest average loglikelihood
     * (where all candidates are trained for the same test string)
     */
    public ModelMatcher getBestMatch(ArrayList<ModelMatcher> candidates)
    {
        ModelMatcher max = candidates.get(0);
        double maxi = candidates.get(0).getAverageLogLikelihood();
        for(ModelMatcher dummy : candidates ){
            if(dummy.getAverageLogLikelihood()  > maxi ){
                maxi = dummy.getAverageLogLikelihood();
                max = dummy;
            }
        }
        return max;
    }

    /** @return String an *explanation* of
     * why the test string is the match from the candidate models
     */
    public String explainBestMatch(ModelMatcher best) {

        String s = "This is the best match because it \n" +
            "has the greatest log likelihood compared to the \n" +
            "others" + " its loglikelihood is : " + best.getAverageLogLikelihood();

        return s;
    }

    /** Display an error to the user in a manner appropriate
     * for the interface being used.
     *
     * @param message
     */
    public void displayError(String message) {
        // LEAVE THIS METHOD EMPTY
    }

}
```

ProjectTest.java

```java
import static org.junit.Assert.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import java.util.HashSet;
import java.util.Set;
import java.util.Collections;
import java.util.ArrayList;

/**
 * The test class ProjectTest for student test cases.
 * Add all new test cases to this task.
 *
 * @author  Joshua Ng (20163079), Mohamed Yusuf (22273364)
 * @version (24/5/2017)
 */
public class ProjectTest
{
    /**
     * Default constructor for test class ProjectTest
     */
    public ProjectTest()
    {
    }

    /**
     * Sets up the test fixture.
     *
     * Called before every test case method.
     */
    @Before
    public void setUp()
    {
    }

    /**
     * Tears down the test fixture.
     *
     * Called after every test case method.
     */
    @After
    public void tearDown()
    {
    }

    //TODO add new test cases from here include brief documentation

    @Test(timeout=1000 , expected = IllegalArgumentException.class)
    public void testSensibleToStringSize() {
        MarkovModel m = new MarkovModel(2,"c");
        MarkovModel k = new MarkovModel(2,null);
        MarkovModel l = new MarkovModel(2,"");
    }

    @Test(timeout=1000)
    public void testGetDistinctNgrams() {
        NgramAnalyser ngramAna1 = new NgramAnalyser(2, "123");
        Set<String> s1 = ngramAna1.getDistinctNgrams();

        //Set<String> someSet = new HashSet<String>();
        ArrayList<String> a1 = new ArrayList<String>(s1);
        Collections.sort(a1);

        ArrayList<String> a2 = new ArrayList<String>();
        a2.add("12");
        a2.add("23");
        a2.add("31");
        assertEquals(a2, a1);
    }

    @Test(timeout=1000)
    public void testLaplaceExample() {
        MarkovModel markovMo2 = new MarkovModel(2, "aabcabaacaac");
        assertEquals(0.5000, markovMo2.laplaceEstimate("aac"), 0.0001);
        assertEquals(0.1667, markovMo2.laplaceEstimate("aaa"), 0.0001);
        assertEquals(0.3333, markovMo2.laplaceEstimate("aab"), 0.0001);
    }

    @Test(timeout=1000)
    public void testSimpleExample() {
      MarkovModel markovMo1 = new MarkovModel(2, "aabcabaacaac");
        assertEquals(0.33333, markovMo1.simpleEstimate("aab"), 0.0001);
    }

    @Test
    public void testTask3example()
    {
        MarkovModel model = new MarkovModel(2, "aabcabaacaac");
        ModelMatcher modelMat1 = new ModelMatcher(model, "aabbcaac");
        assertEquals(-0.3848976, modelMat1.getAverageLogLikelihood(), 0.0001);
        assertEquals(-0.30102999, modelMat1.getLogLikelihood("bca"), 0.0001);
    }

    @Test
    public void matchControllerTest()
    {
        java.util.ArrayList<java.lang.String> arrayLis1 = new java.util.ArrayList<>();
        arrayLis1.add("aaaaaaaaaaaaaaaaaacaaaaaaaaaacccaaaaaaacaaaaaaaaaaaaaaaaaaaaaaa");
        arrayLis1.add("bbbbbbbbbbbbbbbbbbcbbbbcccbbbbbbbbbbbbbbbbbbbbbbbbbcbbbbbbbbbbb");
        arrayLis1.add("cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc");
        arrayLis1.add("ddddddddddddddddddccdddddddddddddddddcccdddddddddddcdddddddddd");
        MatcherController matcherC1 = new MatcherController(2, arrayLis1, "ccc");
        java.util.ArrayList<ModelMatcher> arrayLis2 = matcherC1.matcherList;
        ModelMatcher modelMat1 = matcherC1.getBestMatch(arrayLis2);
        assertEquals(arrayLis2.get(2), matcherC1.getBestMatch(arrayLis2));
    }


}
```