**Code**:

```
import java.util.*;
import java.io.File;
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.FileNotFoundException;


public class Main {

	//if boolean is true its a spam email, otherwise its ham
	static HashMap<ArrayList<String>,Boolean> test=new HashMap<ArrayList<String>,Boolean>();

	//test is split into the trainSet and testSet
	static HashMap<ArrayList<String>,Boolean> trainSet=new HashMap<ArrayList<String>,Boolean>();
	static HashMap<ArrayList<String>,Boolean> testSet=new HashMap<ArrayList<String>,Boolean>();

	//spamDict
	//the key is the word and the value is the occurrence
	static HashMap<String, Integer> spamWords = new HashMap<String, Integer>();

	//HamDict
	static HashMap<String, Integer> hamWords = new HashMap<String, Integer>();

	public static void main(String args[]) {

		int spamEmails = readFilesProcess("enron1/spam", true);

		int hamEmails = readFilesProcess("enron1/ham", false);

		int totalEmails=spamEmails+hamEmails;

		//split into train and test sets
		//no need for random shuffle as hashmap stores elements randomly
		int splitTerm=(int) (0.8*test.size());

		int i=0;
		for(Map.Entry<ArrayList<String>, Boolean> e : test.entrySet()) {
			if(i++<=splitTerm)
				trainSet.put(e.getKey(), e.getValue());
			else
				testSet.put(e.getKey(), e.getValue());
		}

		//Construct dictionaries using train set
		constructDictionaries();

		//tests the testSet
```

```
		testEmails();
	}

	//reads the files in the spam/ham folder, does pre-processing and fills out the test hashMap.
	//returns the number of files in the folder(spam/ham) being read
	public static int readFilesProcess(String location,Boolean sH) {

		File folder = new File(location);
		File[] listOfFiles = folder.listFiles();

		//Stores the file name and location
		String fileName = "";

		//Stores the given email as a whole
		String wholeText = "";

		BufferedReader br;
		StringBuilder sb;

		//Iterates through all the files in the folder
		for (File file : listOfFiles) {
			if (file.isFile()) {
				//Gets the exact location of each file
				fileName = location + "/" + file.getName();
				try {
					br = new BufferedReader(new FileReader(fileName));
					sb = new StringBuilder();
					//Reads from the file line by line
					String line = br.readLine();

					while (line != null) {
						//Reads from the file line by line until the end of file
						sb.append(line);
						//and appends it to the string builder
						line = br.readLine();
					}
					//makes the input text a string
					wholeText = sb.toString();
					br.close();

					//Pre-processing

					//1) Normalize 2) Tokenize
					//This first removes all non-letter characters, folds to lowercase, then
splits the input,
					//doing all the work in a single line:
					String[] splitText = wholeText.replaceAll("[^a-zA-Z ]",
"").toLowerCase().split("\\s+");

					//4) Filter StopWords
```

```
                                        ArrayList<String> stopWords = new ArrayList<>();
                                        try (BufferedReader br2 = new BufferedReader(new
FileReader("stopwordsNLP.txt"))) {
                                            while (br2.ready()) {
                                                stopWords.add(br2.readLine());
                                            }
                                        }

                                        ArrayList<String> splitText2=new ArrayList<String>();

                                        for(int i=0;i<splitText.length;++i) {
                                                if(stopWords.contains(splitText[i])) {
                                                        continue;
                                                }
                                                splitText2.add(splitText[i]);
                                        }

                                        test.put(splitText2,sH);
                                }
                                catch (FileNotFoundException e) {
                                        e.printStackTrace();
                                }
                                catch (IOException e) {
                                        e.printStackTrace();
                                }
                        }
                }
                return listOfFiles.length;
        }

        public static void constructDictionaries() {
                for(Map.Entry<ArrayList<String>, Boolean> e : trainSet.entrySet()) {
                        //(e.getKey(), e.getValue());
                        boolean trigger=e.getValue();

                        if(trigger) {

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

                                ArrayList<String> splitText2=e.getKey();
                                for (String ss : splitText2) {
                                        if (ss.length() > 0) {
                                                if(usedWords.contains(ss) == false) {
                                                        usedWords.add(ss);
                                                insertIntoDict(ss,spamWords);

                                                }
                                        }
                                }
                        }
```

```
                else {
                        ArrayList<String> usedWords = new ArrayList<String>();
                        ArrayList<String> splitText2=e.getKey();
                        for (String ss : splitText2) {
                                if (ss.length() > 0) {
                                        if(usedWords.contains(ss) == false) {
                                                usedWords.add(ss);
                                        insertIntoDict(ss,hamWords);

                                        }
                                }
                        }
                }
        }
        return;
}

public static void insertIntoDict(String word, HashMap<String, Integer> mapH) {
        int count = 0;

        if (mapH.get(word) != null) {
                count = mapH.get(word);
                count++;
                mapH.put(word, count);
        }
        else {
                mapH.put(word,1);
        }
}

public static void testEmails() {

        //go through testSet and count number of spam and ham emails
        double actualSpamCount=0;
        double actualHamCount=0;

        boolean isSpam;
        int spamCounter = 0;


        for(Map.Entry<ArrayList<String>, Boolean> e : testSet.entrySet()) {
                if(e.getValue()){
                        actualSpamCount++;
                }
                else{
                        actualHamCount++;
                }
        }
```

```
		int emailCount = 0;
		for(Map.Entry<ArrayList<String>, Boolean> e : testSet.entrySet()) {
			isSpam = testSpamHam(e.getKey(),actualSpamCount,actualHamCount);

			if (isSpam)
				spamCounter++;

			emailCount++;
		}

		System.out.println("Number of spams in the set: " + spamCounter);

		int hamCounter=emailCount-spamCounter;
		System.out.println("Number of hams in the set: " + hamCounter);

		System.out.println("Actual spams in the set: " + actualSpamCount);
		System.out.println("Actual hams in the set: " + actualHamCount);

		return;
	}

	//Calculates P(Spam|x), P(Ham|x) and returns true if the email is spam and false if it isn't
	public static boolean testSpamHam(ArrayList<String> email,double actualSpamCount,double
actualHamCount){
		double probOfSpam = actualSpamCount/(actualSpamCount+actualHamCount);
		double probOfHam =  actualHamCount/(actualSpamCount+actualHamCount);

		double spam_x;
		double ham_x;

		//calculate sum(SpamDict.values())
		double spamSum=0;
		for(Map.Entry<String, Integer> e : spamWords.entrySet()) {
			spamSum+=e.getValue();
		}

		//calculate sum(HamDict.values())
		double hamSum=0;
		for(Map.Entry<String, Integer> e : hamWords.entrySet()) {
			hamSum+=e.getValue();
		}

		//Each of these counts the number of times each word appeared in the ham/spam set
		String word;

		double totalExpSpam=0.0;
		double totalExpHam=0.0;
		//For each word in the email
		for (int i=0;i<email.size();++i) {
			word=email.get(i);
```

```
                        //calculate P(word|Spam)
                        if(spamWords.get(word)!=null) {
                                //(spamWords.get(word)+1)/sum(spamdict.values())+len(spamDict)+1
                                double numerator=spamWords.get(word)+1.0;
                                double denominator=spamSum+spamWords.size()+1.0;
                                totalExpSpam+=Math.log(numerator/denominator);
                        }
                        else {
                                double numerator=0+1.0;
                                double denominator=spamSum+spamWords.size()+1.0;
                                totalExpSpam+=Math.log(numerator/denominator);
                        }

                        if(hamWords.get(word)!=null) {
                                //(spamWords.get(word)+1)/sum(spamdict.values())+len(spamDict)+1
                                double numerator=hamWords.get(word)+1.0;
                                double denominator=hamSum+hamWords.size()+1.0;
                                totalExpHam+=Math.log(numerator/denominator);
                        }
                        else {
                                double numerator=0+1.0;
                                double denominator=hamSum+hamWords.size()+1.0;
                                totalExpHam+=Math.log(numerator/denominator);
                        }
                }

                spam_x=Math.log(probOfSpam)+(totalExpSpam);
                ham_x=Math.log(probOfHam)+(totalExpHam);

                //checks which probability is higher and then determines whether the email is spam or not
                if (spam_x > ham_x) {
                        return true;
                }
                else {
                        return false;
                }
        }
}
```

**Explanation**:
This is a java implementation of the Naive Bayes Classifier that follows the steps as shown on the writeup. Since Java has no support for NLTK, I was not able to implement a lemmatizer for pre-processing, although I performed all other pre-processing steps such as normalize, tokenize, stopwords. I believe my accuracy would improve if a lemmatizer was implemented as it would improve my classification.

**Results**:

Average Results:
Predicted number of spams in the set: 309
Predicted number of hams in the set: 633
Actual number of spams in the set: 325
Actual number of hams in the set: 617

Confusion Matrix:
True Positive: 309
True Negative: 617
False Positive: 0
False Negative: 16
Accuracy: 0.7205

| I | | Truth | |
|---|---|---|---|
| | | Spam | Ham |
| Predicted | Spam | 309 | 0 |
| | Ham | 16 | 617 |