EVANS'S DIEGO PARSER

Part 2

Abstract

A program to parse a conversation in Diego, a language with a defined alphabet and grammar, and output who is and is not a speaker of the language.

Ethan Evans

ewevans@umich.edu

Contents

1. Problem Statement

In order to find impersonators of the Diego language, defined below, a program will be created to accept a conservation as input and output a report of each person's words (including if they are valid or not).

Diego is a simple language with a small alphabet, which is defined as:

$$\Sigma = \{\mathrm{a, b, d, s}\}, \text{where } 's' \text{ is a space}$$

The grammar of Diego consists of stops, plosives, syllables, words, and sentences, defined as:

$< \mathrm{stop} > \rightarrow \mathrm{b} \mid \mathrm{d}$

$< \mathrm{plosive} > \rightarrow < \mathrm{stop} > \mathrm{a}$

$< \mathrm{syllable} > \rightarrow < \mathrm{plosive} > \mid < \mathrm{plosive} >< \mathrm{stop} > \mid \mathrm{a} < \mathrm{plosive} > \mid \mathrm{a} < \mathrm{stop} >$

$< \mathrm{word} > \rightarrow < \mathrm{syllable} > \mid < \mathrm{syllable} >< \mathrm{word} >< \mathrm{syllable} >$

$< \mathrm{sentence} > \rightarrow < \mathrm{word} > \mid < \mathrm{sentence} > \mathrm{s} < \mathrm{word} >$

Conservations will input as a text file named "CIS400A1.dat" with the following form on each line:

$< Name\ of\ person\ speaking >: < sentence\ in\ Diego >$

For example, a sample conservation can be seen below:

Colette: ba ababadada bad dabbada

Megan: abdabaadab adaba

Lynn: dad ad abaadad badadbaad

Output of the program will include evaluation of each word (word or not a word). Entire sentences will be evaluated. All exceptions will be accounted for including empty file, invalid conversation, no file, etc.

2. Requirements

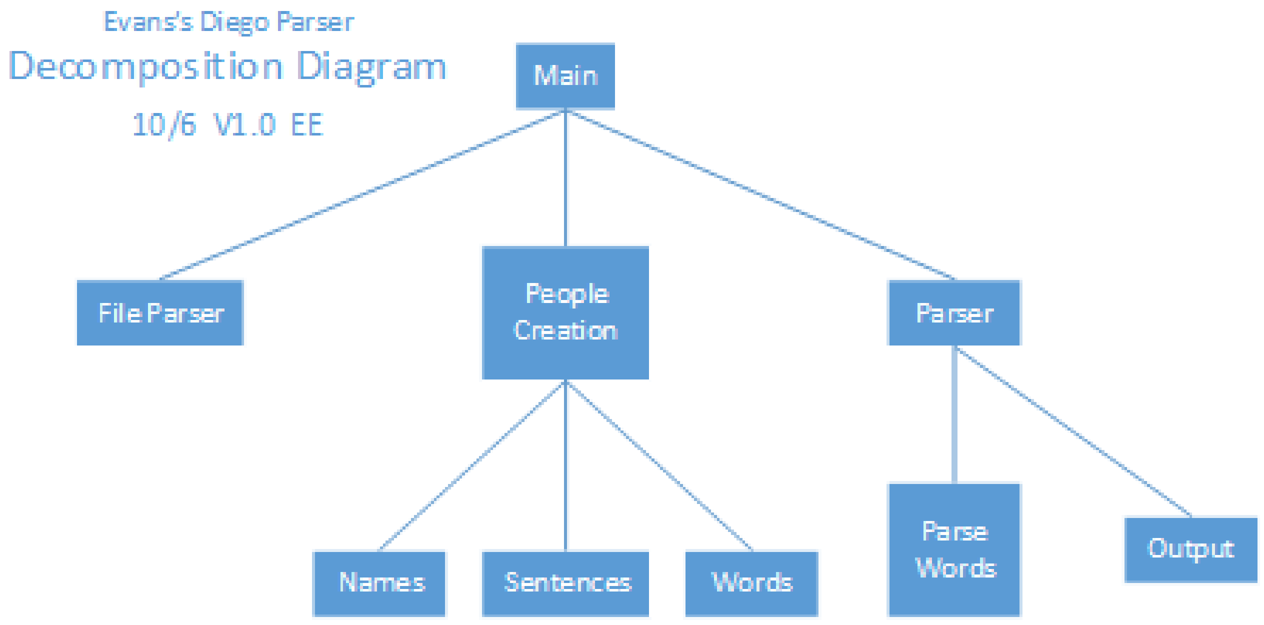
2.1 Assumptions

- The only input will come from the file called “CIS400A1.dat.”
- If the conversation is invalid, an error message will instruct the user to fix the format.

2.2 Specifications

- The program will open the file “CIS400A1.dat”
    - If the file is empty, the user will be notified that the file is empty
    - If the file doesn’t exist, the user will be notified that the file doesn’t exist.
- The program will evaluate the conversation in the file
    - If the conservation is in an unusable format, the user will be notified with the correct format.
    - If the conversation is in a correct format, the program will parse each sentence to check if it is correct Diego.
        - Each person’s speech will be output at a time, indicating they speak Diego with “Found out Diego secrets” and “Impersonator caught” if one of the words spoken by the person is not valid.
        - Each person’s speech is broken down into words, which are individually listed with “word” if the word is valid, and “not a word” if it is not on separate lines.
    - The full sentence will be evaluated even if a word in the beginning is found to be invalid Diego speech.

3. Decomposition Diagram



4. Test Strategy

- File Input
    - Files will need to be checked for existence and non-emptiness
- Person Creation
    - The conversation input will need to be checked for formatting errors
- Correct Parsing
    - Testing of many different words in Diego will be completed.
- Output
    - Ensure correct output and output formatting.

5. Test Plan Version 1

Test Strategy	Test Number	Description	Input	Expected Output	Actual Output	Pass/Fail
File Input	1	No Input File				
File Input	2	Input file is empty				
Person Creation	3	Invalid conversation: No colon on line				
Person Creation	4	Verify Persons are created for each line of valid speech				
Person Creation	5	Person talking on different lines are attributed to same person				
Correct Parsing	6	Test all small words in Diego (2-3 char length)				
Correct Parsing	7	Test large words in Diego				
Correct Parsing	8	Test large words that aren’t in Diego				
Correct Paring	9	Test words are evaluated after a “not a word”				
Output	10	Impersonators and Diego speakers are correctly labeled				

6. Initial Algorithm

1. Create function to create string array from text file, where each line of text is an element of the array.
    - a. If the file does not exist, output to the user “File “CIS400A1.dat” does not exist.”
    - b. If the file is empty, output to the user “File is empty.”
2. Create a class called “Person” that will store the lines of text spoken by that person and a list of words spoken by that person.
    - a. Create a method called “CreatePersons” that will separate the conservation (string array) into multiple Person instances with their speech included in each.
    - b. Create a method that will separate the person’s sentences into words.
3. Create Parsing methods
    - a. Create an int variable called “cursor” that holds our cursor position in the word
    - b. Create an int variable called “savedCursor” that saves a cursor position for backtracking
    - c. Create a method called “lex()” that gets the next character,
        - i. nextToken = array[cursor]
        - ii. increment cursor
        - iii. return nextToken
    - d. Create a method called “term(expected)” to handle terminals
        - i. Return (lex() == expected)
    - e. Create a method for Word()
        - i. Print entering word method
        - ii. SaveCursor()
        - iii. Call Word1()
            - 1. If return 0, cursor = savedCuror, saveCursor(), call Word2()
        - iv. Print exiting word method
        - v. If Word2() returns 1, return 1
        - vi. Else return 0
    - f. Create a method for Word1()
        - i. Print entering Word1
        - ii. Call Syllable()
        - iii. Print exiting Word1
        - iv. If returns 1, return 1
        - v. Else return 0
    - g. Create a method for Word2()
        - i. Print entering Word2
        - ii. Call Syllable()
        - iii. If returns 0, Print exiting Word2, return 0
        - iv. Call Word()
        - v. If returns 0, Print exiting Word2, return 0
        - vi. Call Syllable()
        - vii. If returns 0, Print exiting Word2, return 0
        - viii. Print exiting Word2

```
        ix.   Return 1
h.  Create a method for Syllable
        i.    Print entering syllable method
        ii.   SaveCursor()
        iii.  Call Syllable1()
        iv.   If Syllable1() returns 0, cursor = savedCuror, SaveCursor(), Call Syllable2()
                  1.  Else Print exiting syllable method, return 1
        v.    If Syllable2() returns 0, cursor = savedCuror, SaveCursor(), Call Syllable3()
                  1.  Else Print exiting syllable method, return 1
        vi.   If Syllable3() returns 0, cursor = savedCuror, SaveCursor(), Call Syllable4()
                  1.  Else Print exiting syllable method, return 1
        vii.  If Syllable4() returns 0, return 0
                  1.  Else Print exiting syllable method, return 1
i.  Create a method for Syllable1()
        i.    Print Entering Syllable1
        ii.   Call Plosive()
        iii.  Print Exiting Syllable1
        iv.   If returns 1, return 1
        v.    Else return 0
j.  Create a method for Syllable2()
        i.    Print Entering Syllable2
        ii.   Call Plosive()
                  1.  If returns 0, Print exiting Syllable2, return 0
        iii.  else Call Stop()
                  1.  If returns 0, Print Exiting Syllable2, return 0
                          i.  Else return 0
        iv.   Else return 0
k.  Create a method for Syllable3()
        i.    Print Entering Syllable3
        ii.   If term(a) returns 0, Print exiting Syllable3 method, return 0
        iii.  Call Plosive()
                  1.  If returns 0, Print exiting Syllable3, return 0
        iv.   Else Print exiting Syllable3, return 1
l.  Create a method for Syllable4()
        i.    Print Entering Syllable4
        ii.   If term(a) returns 0, Print exiting Syllable4 method, return 0
        iii.  Call Stop()
                  1.  If returns 0, Print exiting Syllable4, return 0
        iv.   Else Print exiting Syllable4, return 1
m.  Create a method for Plosive()
        i.    Print entering Plosive
        ii.   Call Stop()
        iii.  If Stop returns 0, return 0
        iv.   If term(a) returns 0, Print exiting plosive method, return 0
        v. Print exiting Plosive
        vi. Return 1
    n. Create a method for Stop()
        i. Print entering Stop
        ii. saveCursor()
        iii. Call Stop1()
        iv. If Stop1() returns 0, cursor = savedCuror, saveCursor(), Call Stop2()
        v. If Stop2() returns 0, return 0
        vi. Print Exiting Stop
        vii. Return 1
    o. Create a method for Stop1()
        i. Print Entering Stop1
        ii. If term(b) returns 0, Print exiting Stop1 method, return 0
        iii. Else
            1. Print exiting Stop1 method, return 1
    p. Create a method for Stop2()
        i. Print Entering Stop2
        ii. If term(d) returns 0, Print exiting Stop2 method, return 0
        iii. Else
            1. Print exiting Stop2 method, return 1
4. Create a method to loop through each Person's words and output the parsing results
    a. For each Person in Person[],
        i. For each word in Words[],
            1. Call parse(word)
            2. Store each word and if it is a word or not based on the parse
               method
        ii. Print Speech from <Person name>:
            1. If all words pass the parse, print Found out Diego secrets
            2. Else print Impersonator caught
        iii. For each word in Words[],
            1. Print word
            2. Print word if the word is a word or not a word
```

7. Test Plan Version 2

Test Strategy	Test Number	Description	Input	Expected Output	Actual Output	Pass/Fail
File Input	1	No Input File	None	"Error. No input file, CIS400A1.dat found"		
File Input	2	Input file is empty	Blank file "CIS400A1.dat"	"Error. Input file blank."		
Person Creation	3	Invalid conversation: No colon on line	CIS400A1.dat: Ethan ; dad ab ab	"Error. Conversation does not follow correct format: <Name>: <Speech>		
Person Creation	4	Verify Persons are created for each line of valid speech	CIS400A1.dat: Tom: dadbaba Christina: bababa	Tom "found…" dadbaba word Christina "found o.." bababa word		
Person Creation	5	Person talking on different lines are attributed to same person	CIS400A1.dat: Steve: bab da Steve: dad	Steve "found…" bab word da word dad word		
Correct Parsing	6	Test all small words in Diego (2-3 char length)	CIS400A1.dat: A: ab ad ba da bab bad dab dad aba ada	A "found…" ab word ad word ba word da word (…) word		
Correct Parsing	7	Test large words in Diego	CIS400A1.dat: A: babababababababa baba	A "found…" babababababababab a word		
Correct Parsing	8	Test large words that aren't in Diego	CIS400A1.dat: A: bababdadbababa	A "Impersonator…" bababdadbababa not a word		
Correct Paring	9	Test words are evaluated	CIS400A1.dat: A: abaa dad	A "Impersonator…" abaa not a word dad word		

		after a “not a word”				
Output	10	Impersonators and Diego speakers are correctly labeled	CIS400A1.dat: Tom: dadbaba Christina: babababaa	Tom “found…” dadbaba word Christina “Impersonator…” babababaa not a word		

8. Code

Main.java

```
package com.company;
import java.io.*;
import java.nio.file.*;
import java.util.ArrayList;
import java.util.List;

//Pre: Input via CIS400A1.dat in format of lines of
//          <name> : <sentence>
//
//Post: If valid data,
//   <Persons Name>   Found out Diego secrets or Impersonator Caught
//   <List of sentences spoken>
//   <List of words>   <word or not a word>

public class Main {

    //List of Persons that hold the words and sentences they speak
    static List<Person> people = new ArrayList<>();

    public static void main(String[] args) {


        // write your code here
        String[] conversation;

        //Get lines of text from file
        conversation = QueryFile();


        if (conversation == null) {
            //file error
            //error output already occurred
        } else {
            PeopleParser peopleParser = new PeopleParser();
            peopleParser.CreatePersons(conversation);


            //error output already occurred
            if (people == null) {
                //System.out.print("ERROR: people is null");
                //people is null if invalid format is found

            } //If no lines of text
            else if (people.size() == 0)
            {
                System.out.print("Input file CIS400A1.dat is empty."
                        + "\nConversations are lines of text that follow this form:"
                        + "\n   <person>:<white space><sentence>");
            }
            else
            {   //output to console
                peopleParser.InterrogatePersons();
            }
        }
    }

    //Opens data file and returns lines of text as array
    private static String[] QueryFile ()
    {
        //create file path for CIS400A1.dat
        Path path = Paths.get("CIS400A1.dat");

        try {
            //Get list of lines from text file
            List<String> lines = Files.readAllLines(path);

            //Convert to string array
            String[] arr = lines.toArray(new String[lines.size()]);

            return arr;
        }
        catch (IOException ex)
        {
            System.out.print("File 'CIS400A1.dat' not found.\n");

            return null;
        }
    }
}
```

Person.java

```
package com.company;
import java.util.ArrayList;
import java.util.List;

/** Person
 *   Data structure to pair person with speech
 */
public class Person
{
    //name of speaker who may or may not be a Diego secret keeper
    String name;

    //every line of text from the person
    List<String> sentences = new ArrayList<>();

    //An in-order list of words found in all sentences
    List<String> words = new ArrayList<>();
}
```

PeopleParser.java

```
package com.company;

import java.util.List;
```

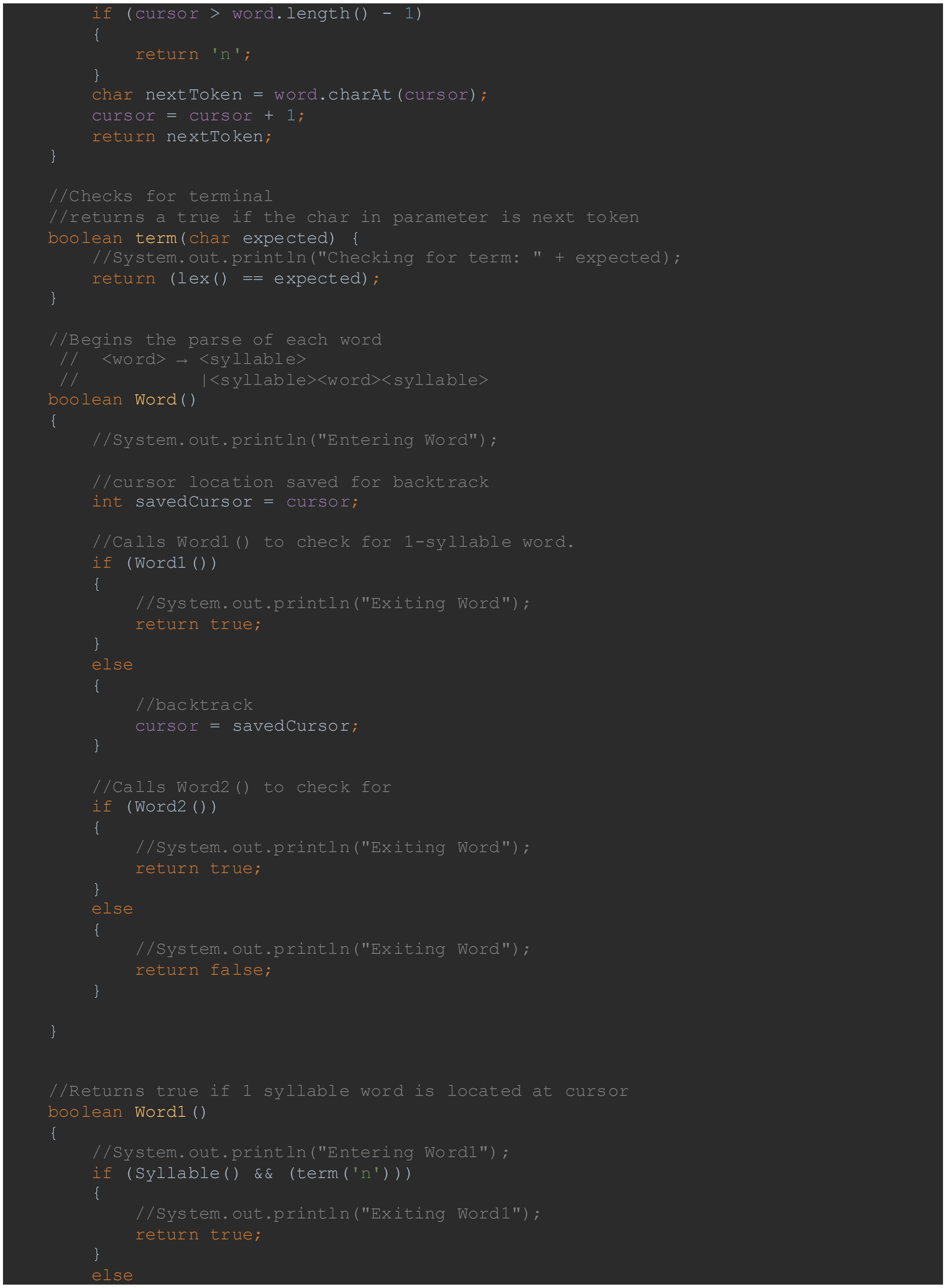
```java
/**
 * Creates List of Persons from lines of text
 * Interrogates Persons to output to console
 *
 */
public class PeopleParser
{

    //C
    public List<Person> CreatePersons(String[] conversation) {

        if (conversation == null)
        {
            System.out.print("File empty. Please add some conversation."
                    + "\nConversations are lines of text that follow this form:"
                    + "\n    <person>:<white space><sentence>");
            return null;
        }
        else
        {
            //counter used to place in person array
            int personCounter = 0;

            //take every line and make a new person if new name
            for (String line : conversation)
            {
                //CHECK for blank line
                if (line.length() != 0)
                {
                    boolean nameExists = false;

                    //remove name and colon, rest is person's sentence
                    String[] lineSplit = line.split(":");

                    //CHECK for too many or too little colons
                    if (lineSplit.length != 2)
                    {
                        System.out.print("Incorrect line format"
                                + "\nConversations are lines of text that follow this
form:"
                                + "\n    <person>:<white space><sentence>");
                        Main.people = null;
                        return null;
                    }
                    String name = lineSplit[0];
                    String sentence = lineSplit[1];
                    Person person = new Person();

                    //if name is in use, set as existing
                    if (Main.people != null) {
                        for (Person tempPerson : Main.people)
                        {
                            if (name.equals(tempPerson.name))
                            {
                                person = tempPerson;
                                nameExists = true;
                            }
                        }


                        //create new person to have new line
                        if (nameExists == false)
                        {
```

```java
                        //increase person count to add person
                        personCounter = personCounter + 1;

                        person.name = name;
                        person.sentences.add(sentence);
                        String[] words = sentence.split("\\s+");
                        for (String word : words)
                        {
                            if (word.length() > 0)
                            {
                                person.words.add(word);
                            }
                        }
                        Main.people.add(person);
                    }
                    else //adding sentence to existing person
                    {
                        person.sentences.add(sentence);
                        String[] words = sentence.split("\\s+");
                        for (String word : words)
                        {
                            if (word.length() > 0)
                                person.words.add(word);
                        }
                    }
                }
            }
        }

        return Main.people;
    }
}

public void InterrogatePersons()
{
    Parser parser = new Parser();

    for (Person person : Main.people)
    {
        boolean speaksDiego = true;

        for (String word : person.words)
        {
            if (parser.Parse(word) == false)
            {
                speaksDiego = false;
            }
        }
        if (speaksDiego)
        {
            System.out.print(person.name + "   Found out Diego secrets\n");
        }
        else
        {
            System.out.print(person.name + "   Impersonator caught\n");
        }
        for (String sentence : person.sentences)
        {
            if (sentence != null)
                System.out.println(sentence);
        }

        System.out.print("   Words:\n");
```

```
            for (String word : person.words)
            {
                System.out.print(word + "            ");
                if (parser.Parse(word))
                {
                    System.out.print("word");
                }
                else
                {
                    System.out.print("not a word");
                }
                System.out.print("\n");
            }

            System.out.print("\n");
        }
    }
}
```

Parser.java

```
package com.company;

/**
 * Parser
 *  The recursive-descent parser for the Diego language
 *
 *  The Diego Language
 *
 *  <stop>  → b
 *             |d
 *  <plosive>  → <stop>a
 *  <syllable> → <plosive>
 *                |<plosive><stop>
 *                |a<plosive>
 *                |a<stop>
 *  <word> → <syllable>
 *           |<syllable><word><syllable>
 *  <sentence> → <word>
 *                |<sentence>s<word>

 */
public class Parser {
    //location of the cursor during the parse
    //Used in pair with savedCursor to control backtracking
    int cursor = 0;

    //word being parsed
    String word;

    //Parses word in Diego and returns true if correct Diego, false otherwise
    // letters not in Diego alphabet return false
    public boolean Parse(String wordInput) {
        cursor = 0;
        word = wordInput;
        return Word();
    }

    //Gives the next token in the word
    // returns 'n' if end of the word
    char lex() {
```



```
        if (cursor > word.length() - 1)
        {
            return 'n';
        }
        char nextToken = word.charAt(cursor);
        cursor = cursor + 1;
        return nextToken;
    }

    //Checks for terminal
    //returns a true if the char in parameter is next token
    boolean term(char expected) {
        //System.out.println("Checking for term: " + expected);
        return (lex() == expected);
    }

    //Begins the parse of each word
     //  <word> → <syllable>
     //           |<syllable><word><syllable>
    boolean Word()
    {
        //System.out.println("Entering Word");

        //cursor location saved for backtrack
        int savedCursor = cursor;

        //Calls Word1() to check for 1-syllable word.
        if (Word1())
        {
            //System.out.println("Exiting Word");
            return true;
        }
        else
        {
            //backtrack
            cursor = savedCursor;
        }

        //Calls Word2() to check for
        if (Word2())
        {
            //System.out.println("Exiting Word");
            return true;
        }
        else
        {
            //System.out.println("Exiting Word");
            return false;
        }

    }


    //Returns true if 1 syllable word is located at cursor
    boolean Word1()
    {
        //System.out.println("Entering Word1");
        if (Syllable() && (term('n')))
        {
            //System.out.println("Exiting Word1");
            return true;
        }
        else
        {
            //System.out.println("Exiting Word1");
            return false;
        }
    }


    // Due to <plosive> and <plosive><stop> both being valid
    // must check different combinations of word lengths for
    //       <word> → <syllable><word><syllable>
    //    is
    //       <word> →<2-char syllable><2-char syllable><word>
    //                |<2-char syllable><3-char  syllable><word>
    //                |<3-char syllable><2-char syllable><word>
    //                |<3-char syllable><3-char syllable><word>
    boolean Word2()
    {
        //System.out.println("Entering Word2");
        int savedCursor = cursor;

        //2,2,word
        if (Syllable2char() && Syllable2char()  && Word())
        {
            //System.out.println("Exiting Word2");
            return true;
        }
        else
        {
            //backtrack
            cursor = savedCursor;
        }

        //2,3,word
        if (Syllable2char() && Syllable3char()  && Word())
        {
            //System.out.println("Exiting Word2");
            return true;
        }
        else
        {
            //backtrack
            cursor = savedCursor;
        }

        //3,2,word
        if (Syllable3char() && Syllable2char()  && Word())
        {
            //System.out.println("Exiting Word2");
            return true;
        }
        else
        {
            //backtrack
            cursor = savedCursor;
        }

        //3,3,word
        if (Syllable3char() && Syllable3char()  && Word())
        {
            //System.out.println("Exiting Word2");
            return true;
        }
        else
        {
            //backtrack
            return false;
        }

    }

    // Returns true if 2 character syllable is located at cursor
    //   <syllable> → <plosive>          <--------
    //                |<plosive><stop>
    //                |a<plosive>
    //                |a<stop>           <--------
    boolean Syllable2char()
    {
        int savedCursor = cursor;
        if (term('a') && Stop())
        {
            return true;
        }
        else
        {
            //backtrack
            cursor = savedCursor;
        }
        if (Plosive())
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    // Returns true if 3 character syllable is located at cursor
    //   <syllable> → <plosive>
    //                |<plosive><stop>      <--------
    //                |a<plosive>           <--------
    //                |a<stop>
    boolean Syllable3char()
    {
        int savedCursor = cursor;
        if (term('a') && Plosive())
        {
            return true;
        }
        else
        {
            //backtrack
            cursor = savedCursor;
        }

        if (Plosive() && Stop())
        {
            return true;
        }
        else
        {
            return false;
        }
    }

     // syllable parsing for single syllables
     //
     //   <syllable> → <plosive>
     //                |<plosive><stop>
     //                |a<plosive>
     //                |a<stop>
    boolean Syllable()
    {
        //System.out.println("Entering Syllable");

        //SaveCursor();

        int savedCursor = cursor;


        if (Syllable2())
        {
            //System.out.println("Exiting Syllable");
            return true;
        }
        else
        {
            //backtrack
            cursor = savedCursor;
        }
        if (Syllable1())
        {
            //System.out.println("Exiting Syllable");
            return true;
        }
        else
        {
            //backtrack
            cursor = savedCursor;
        }
        if (Syllable3())
        {
            //System.out.println("Exiting Syllable");
            return true;
        }
        else
        {
            //backtrack
            cursor = savedCursor;
        }
        if (Syllable4())
        {
            //System.out.println("Exiting Syllable");
            return true;
        }
        else
        {
            //System.out.println("Exiting Syllable");
            return false;
        }
    }

    boolean Syllable1()
    {
        //System.out.println("Entering Syllable1");
        if (Plosive())
        {
            //System.out.println("Exiting Syllable1");
            return true;
        }
        else
        {
            //System.out.println("Exiting Syllable1");
            return false;
        }
    }

    boolean Syllable2()
    {
        //System.out.println("Entering Syllable2");
        if (Plosive() == false)
        {
            //System.out.println("Exiting Syllable2");
            return false;
        }
        if (Stop())
        {
            //System.out.println("Exiting Syllable2");
            return true;
        }
        else
        {
            //System.out.println("Exiting Syllable2");
            return false;
        }
    }

    boolean Syllable3()
    {
        //System.out.println("Entering Syllable3");
        if (term('a') == false)
        {
            //System.out.println("Exiting Syllable3");
            return false;
        }
        if (Plosive())
        {
            //System.out.println("Exiting Syllable3");
            return true;
        }
        else
        {
            //System.out.println("Exiting Syllable3");
            return false;
        }
    }

    boolean Syllable4()
    {
        //System.out.println("Entering Syllable4");
        if (term('a') == false)
        {
           // System.out.println("Exiting Syllable4");
            return false;
        }
        if (Stop())
        {
            //System.out.println("Exiting Syllable4");
            return true;
        }
        else
        {
            //System.out.println("Exiting Syllable4");
            return false;
        }
    }

    //
    //<plosive>  → <stop>a
    //
    boolean Plosive()
    {
        //System.out.println("Entering Plosive");
        if (Stop() == false)
        {
            //System.out.println("Exiting Plosive");
            return false;
        }
        if (term('a'))
        {
            //System.out.println("Exiting Plosive");
            return true;
        }
        else
        {
            //System.out.println("Exiting Plosive");
            return false;
        }
    }

    //   <stop>  → b
    //             |d
    boolean Stop()
    {
        //System.out.println("Entering Stop");
        //SaveCursor();

        int savedCursor = cursor;

        //b
        if (Stop1())
        {
            //System.out.println("Exiting Stop");
            return true;
        }
        else
        {
            //backtrack
            cursor = savedCursor;
        }

        //d
        if (Stop2())
        {
            //System.out.println("Exiting Stop");
            return true;
        }
        else
        {
            //System.out.println("Exiting Stop");
            return false;
        }
    }

    boolean Stop1()
    {
        //System.out.println("Entering Stop1");
        if (term('b'))
        {
            //System.out.println("Exiting Stop1");
            return true;
        }
        else
        {
            //System.out.println("Exiting Stop1");
            return false;
        }
    }

    boolean Stop2()
    {
       // System.out.println("Entering Stop2");
        if (term('d'))
        {
            //System.out.println("Exiting Stop2");
            return true;
        }
        else
        {
            //System.out.println("Exiting Stop2");
            return false;
        }
    }
}
```

9. Updated Algorithm

1. Create function to create string array from text file, where each line of text is an element of the array.
    a. If the file does not exist, output to the user "File "CIS400A1.dat" does not exist."
    b. If the file is empty, output to the user "File is empty."
2. Create a class called "Person" that will store the lines of text spoken by that person and a list of words spoken by that person.
    a. Create a method called "CreatePersons" that will separate the conservation (string array) into multiple Person instances with their speech included in each.
        i. Separate the person's sentences into words.
    b. Create a method called "InterrogatePersons" that will output to the console
        i. For each Person in Person[],
            1. Set speaksDiego to true
            2. For each word in Words[],
                a. Call parse(word)
                b. Set speaksDiego to false
            3. Print the person's name
            4. If speaksDiego is true
                a. If all words pass the parse, print Found out Diego secrets
                b. Else print Impersonator caught
            5. Print each sentence on different lines
            6. For each word in Words[],

        a. Print word
        b. Print word if the word is a word or not a word

3. Create Parsing methods
    a. Create an int variable called "cursor" that holds our cursor position in the word
    b. Create string variable for word being parsed
    ~~c. Create an int variable called "savedCursor" that saves a cursor position for backtracking~~
    d. Create method that starts Parse
        i. Set Cursor to 0
        ii. Set word to the input word
        iii. Call Word()
    e. Create a method called "lex()" that gets the next character,
        i. If end of word, return 'n'
        ii. nextToken = array[cursor]
        iii. increment cursor
        iv. return nextToken
    f. Create a method called "term(expected)" to handle terminals
        i. Return (lex() == expected)
    g. Create a method for Word()
        i. Print entering word method
        ii. Create a savedCursor variable initialized as cursor
        iii. Call Word1()
            1. If return 0, cursor = savedCuror, call Word2()
        iv. Print exiting word method
        v. If Word2() returns 1, return 1
        vi. Else return 0
    h. Create a method for Word1()
        i. Print entering Word1
        ii. Call Syllable()
        iii. Print exiting Word1
        iv. If returns 1, return 1
        v. Else return 0
    i. Create a method for Word2()
        i. Print entering Word2
        ii. Create a savedCursor variable initialized as cursor
        iii. If 2-char syllable, 2-char syllable, word at cursor return true
        iv. Else cursor = savedCursor
        v. If 2-char syllable, 3-char syllable, word at cursor return true
        vi. Else cursor = savedCursor
        vii. If 3-char syllable, 2-char syllable, word at cursor return true
        viii. Else cursor = savedCursor
        ix. If 3-char syllable, 3-char syllable, word at cursor return true
        x. Else return 0
    j. Create method for 2-char syllable
        i. Create a variable savedCursor set as cursor

```
        ii.   If next token is 'a' and followed by Stop() true, return true
        iii.  Else cursor = savedCursor
        iv.   If Plosive() true, return true
        v.    Else return false
k.  Create method for 3-char syllable
        i.    Create a variable savedCursor set as cursor
        ii.   If next token is 'a' and followed by Plosive() true, return true
        iii.  Else cursor = savedCursor
        iv.   If Plosive() true and Stop() true, return true
        v.    Else return false
l.  Create a method for Syllable
        i.    Print entering syllable method
        ii.   Create a savedCursor variable set as cursor
        iii.  Call Syllable1()
        iv.   If Syllable2() returns 0, cursor = savedCuror, SaveCursor(), Call Syllable1()
                  1.  Else Print exiting syllable method, return 1
        v.    If Syllable1() returns 0, cursor = savedCuror, SaveCursor(), Call Syllable3()
                  1.  Else Print exiting syllable method, return 1
        vi.   If Syllable3() returns 0, cursor = savedCuror, SaveCursor(), Call Syllable4()
                  1.  Else Print exiting syllable method, return 1
        vii.  If Syllable4() returns 0, return 0
                  1.  Else Print exiting syllable method, return 1
m.  Create a method for Syllable1()
        i.    Print Entering Syllable1
        ii.   Call Plosive()
        iii.  Print Exiting Syllable1
        iv.   If returns 1, return 1
        v.    Else return 0
n.  Create a method for Syllable2()
        i.    Print Entering Syllable2
        ii.   Call Plosive()
                  1.  If returns 0, Print exiting Syllable2, return 0
        iii.  else Call Stop()
                  1.  If returns 0, Print Exiting Syllable2, return 0
                          i.  Else return 0
        iv.   Else return 0
o.  Create a method for Syllable3()
        i.    Print Entering Syllable3
        ii.   If term(a) returns 0, Print exiting Syllable3 method, return 0
        iii.  Call Plosive()
                  1.  If returns 0, Print exiting Syllable3, return 0
        iv.   Else Print exiting Syllable3, return 1
p.  Create a method for Syllable4()
        i.    Print Entering Syllable4
        ii.   If term(a) returns 0, Print exiting Syllable4 method, return 0
       iii.   Call Stop()
                  1.  If returns 0, Print exiting Syllable4, return 0
        iv.   Else Print exiting Syllable4, return 1
q.  Create a method for Plosive()
         i.   Print entering Plosive
        ii.   Call Stop()
       iii.   If Stop returns 0, return 0
        iv.   If term(a) returns 0, Print exiting plosive method, return 0
         v.   Print exiting Plosive
        vi.   Return 1
r.  Create a method for Stop()
         i.   Print entering Stop
        ii.   Create a savedCursor variable set as cursor
       iii.   Call Stop1()
        iv.   If Stop1() returns 0, cursor = savedCuror, saveCursor(), Call Stop2()
         v.   If Stop2() returns 0, return 0
        vi.   Print Exiting Stop
       vii.   Return 1
s.  Create a method for Stop1()
         i.   Print Entering Stop1
        ii.   If term(b) returns 0, Print exiting Stop1 method, return 0
       iii.   Else
                  1.  Print exiting Stop1 method, return 1
t.  Create a method for Stop2()
         i.   Print Entering Stop2
        ii.   If term(d) returns 0, Print exiting Stop2 method, return 0
       iii.   Else
                  1.  Print exiting Stop2 method, return 1
```

10. Test Plan Version 3

Test Strategy	Test Number	Description	Input	Expected Output	Actual Output	Pass/Fail
File Input	1	No Input File	None	"Error. No input file, CIS400A1.dat found"	File 'CIS400A1.dat' not found.	PASS
File Input	2	Input file is empty	Blank file "CIS400A1.dat"	"Error. Input file blank."	Input file CIS400A1.dat is empty. Conversations are lines of text that follow this form: <person>:<white space><sentence>	PASS
Person Creation	3	Invalid conversation: No colon on line	CIS400A1.dat: Ethan ; dad ab ab	"Error. Conversation does not follow correct format: <Name>: <Speech>	Incorrect line format Conversations are lines of text that follow this form: <person>:<white space><sentence>	PASS
Person Creation	4	Verify Persons are created for each line of valid speech	CIS400A1.dat: Tom: dadbaba Christina: bababa	Tom "found…" dadbaba word Christina "found o.." bababa word	Tom Found out Diego secrets dadbaba Words: dadbaba word Christina Found out Diego secrets bababa Words: bababa word	PASS
Person Creation	5	Person talking on different lines are attributed to same person	CIS400A1.dat: Steve: bab da Steve: dad	Steve "found…" bab word da word dad word	Steve Found out Diego secrets bab da dad Words: bab word da word dad word	PASS

Correct Parsing	6	Test all small words in Diego (2-3 char length)	CIS400A1.dat: A: ab ad ba da bab bad dab dad aba ada	A "found…" ab word ad word ba word da word (…) word	A Found out Diego secrets ab ad ba da bab bad dab dad aba ada Words: ab word ad word ba word da word bab word bad word dab word dad word aba word ada word	PASS
Correct Parsing	7	Test large words in Diego	CIS400A1.dat: A: babababababababa	A "found…" babababababababa word	A Found out Diego secrets babababababababa Words: babababababababa word	PASS
Correct Parsing	8	Test large words that aren't in Diego	CIS400A1.dat: A: bababdadbababaa	A "Impersonator…" bababdadbababa not a word	A Found out Diego secrets bababdadbababa Words: bababdadbababaa not a word	PASS
Correct Paring	9	Test words are evaluated after a "not a word"	CIS400A1.dat: A: abaa dad	A "Impersonator…" abaa not a word dad word	A Impersonator caught abaa dad Words: abaa not a word dad word	PASS

Output	10	Impersonators and Diego speakers are correctly labeled	CIS400A1.dat: Tom: dadbaba Christina: babab aaa	Tom "found…" dadbaba word Christina "Impersonator…" babab aaa not a word	Tom Found out Diego secrets dadbaba Words: dadbaba word Christina Impersonator caught bababaaa Words: bababaaa not a word	PASS
Correct Parsing	11	More parsing tests	See Test Case 11 input below	See Test Case 11 Output below	See Test Case 11 Output below	PASS

Test Case 11 Input:

```
2 or 3 letter SUCC: ab ad da ba bab bad dab dad aba ada
2 or 3 letter FAIL: aa bb dd bd db af be aaa bbb ddd aad aab baa daa
plosive vs plosivestop SUCC: bababa dadada baddabab abababaaba
plosive vs plosivestop FAIL: babbababb dabbababab baabbaa dabbabbaba
big words SUCC: babababababa dabbadabbada abaabaababababa dabbaddabadda
big words FAIL: baddabbada abaabaabaaba abababaddabbaba badbadbadbadbadbad
extreme SUCC: babababababababababababa ababadababababababababababababab
extreme FAIL: dabbaabaabaabaabaabaabaabaaba
```

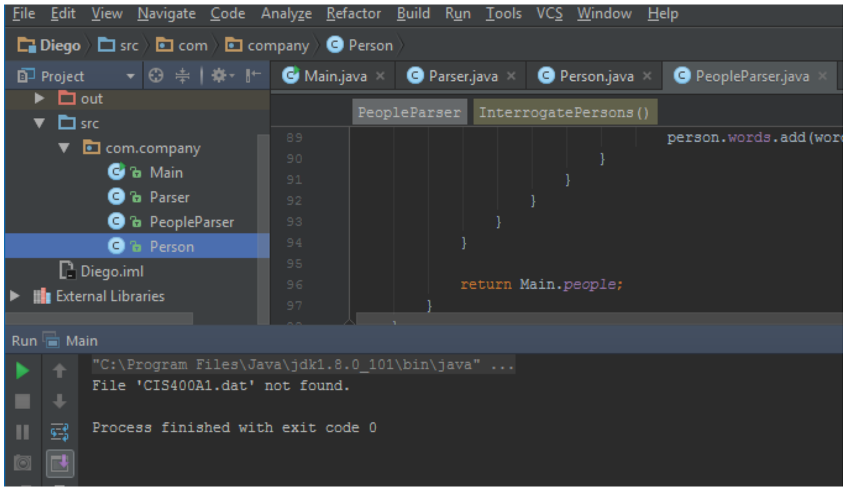
Test Case 11 Output:

```
2 or 3 letter SUCC   Found out Diego secrets
 ab ad da ba bab bad dab dad aba ada
   Words:
ab        word
ad        word
da        word
ba        word
bab        word
bad        word
dab        word
dad        word
aba        word
ada        word

2 or 3 letter FAIL   Impersonator caught
 aa bb dd bd db af be aaa bbb ddd aad aab baa daa
   Words:
aa        not a word
bb        not a word
dd        not a word
bd        not a word
db        not a word
af       not a word
be        not a word
aaa        not a word
bbb        not a word
ddd        not a word
aad        not a word
aab        not a word
baa        not a word
daa        not a word
```

```
plosive vs plosivestop SUCC   Found out Diego secrets
 bababa dadada baddabab ababaaba
   Words:
bababa          word
dadada          word
baddabab          word
ababaaba          word

plosive vs plosivestop FAIL   Impersonator caught
 babbababb dabbababab baabbaa dabbabbaba
   Words:
babbababb          not a word
dabbababab          not a word
baabbaa          not a word
dabbabbaba          not a word

big words SUCC   Found out Diego secrets
 bababababa dabbadabbada abaabaabababa dabbaddabadda
   Words:
bababababa          word
dabbadabbada          word
abaabaabababa          word
dabbaddabadda          word

big words FAIL   Impersonator caught
 baddabbada abaabaabaaba ababaddabbaba badbadbadbadbadbad
   Words:
baddabbada          not a word
abaabaabaaba          not a word
ababaddabbaba          word
badbadbadbadbadbad          not a word

extreme SUCC   Found out Diego secrets
 babababababababababababa ababadabababababababababababab
   Words:
babababababababababababa          word
ababadabababababababababababab          word

extreme FAIL   Impersonator caught
 dabbaabaabaabaabaabaabaabaaba
   Words:
dabbaabaabaabaabaabaabaabaaba          not a word
```

11. Screenshots

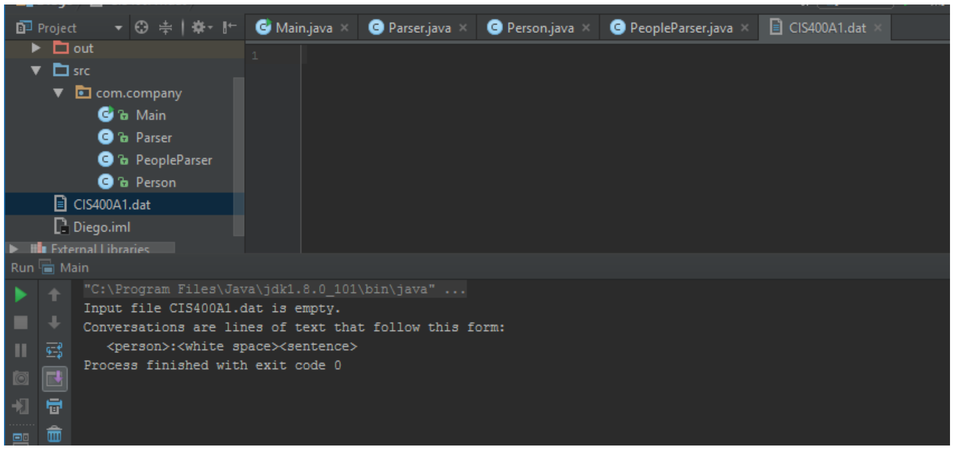
Test Case 1


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Diego src com company Person
Project
Main.java Parser.java Person.java PeopleParser.java
out
src
com.company
Main
Parser
PeopleParser
Person
Diego.iml
External Libraries
PeopleParser InterrogatePersons()
Run Main

```
                                        person.words.add(wor
                                    }
                                }
                            }
                        }
                    }

                    return Main.people;
                }
```

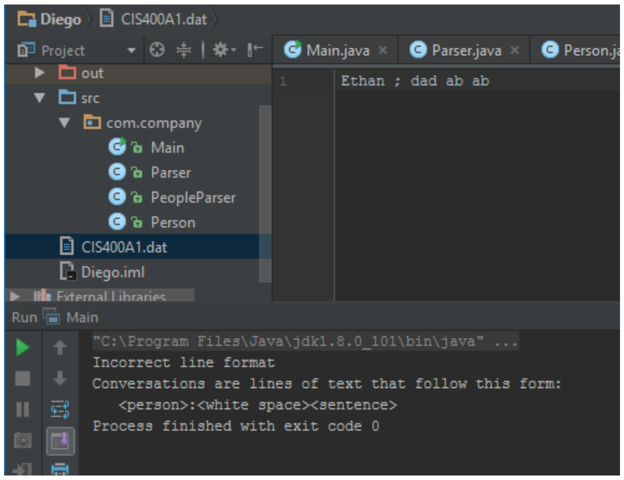
```
"C:\Program Files\Java\jdk1.8.0_101\bin\java" ...
File 'CIS400A1.dat' not found.

Process finished with exit code 0
```

Test Case 2



```
"C:\Program Files\Java\jdk1.8.0_101\bin\java" ...
Input file CIS400A1.dat is empty.
Conversations are lines of text that follow this form:
   <person>:<white space><sentence>
Process finished with exit code 0
```

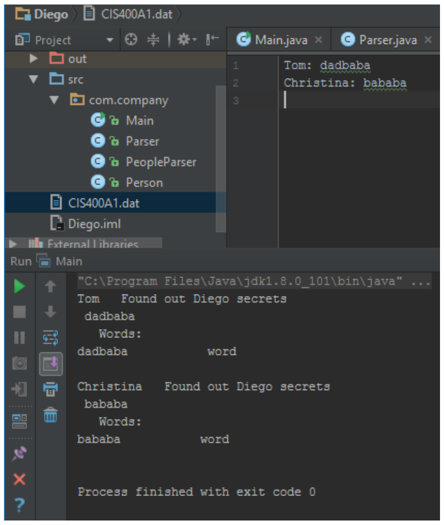
Test Case 3



```
Ethan ; dad ab ab
```

```
"C:\Program Files\Java\jdk1.8.0_101\bin\java" ...
Incorrect line format
Conversations are lines of text that follow this form:
   <person>:<white space><sentence>
Process finished with exit code 0
```

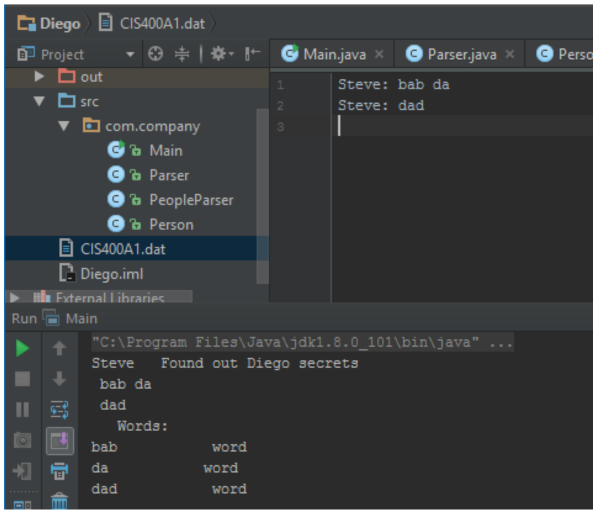
Test Case 4



```
Tom: dadbaba
Christina: bababa
```

```
"C:\Program Files\Java\jdk1.8.0_101\bin\java" ...
Tom   Found out Diego secrets
 dadbaba
   Words:
dadbaba          word

Christina   Found out Diego secrets
 bababa
   Words:
bababa           word


Process finished with exit code 0
```

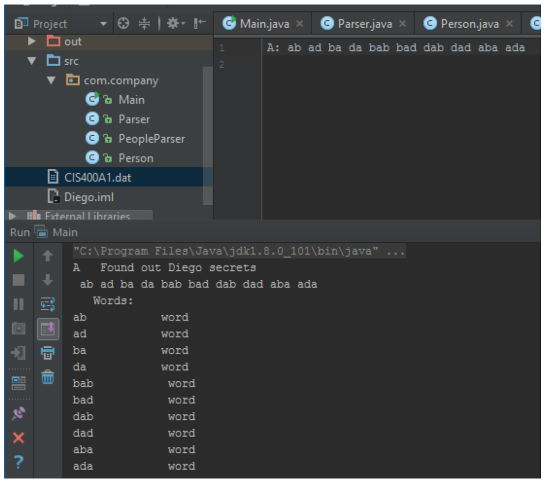
Test Case 5



```
Steve: bab da
Steve: dad
```

```
"C:\Program Files\Java\jdk1.8.0_101\bin\java" ...
Steve   Found out Diego secrets
 bab da
 dad
   Words:
bab            word
da            word
dad            word
```

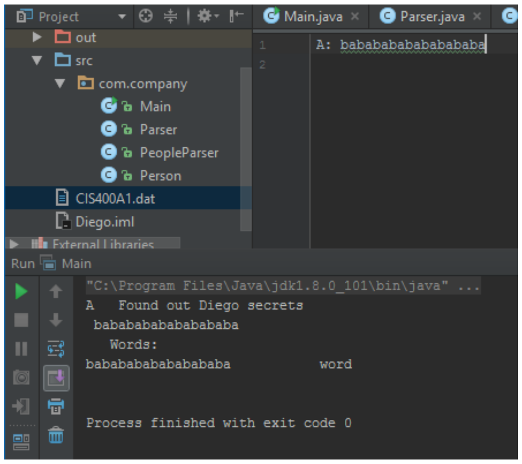
Test Case 6



```
A: ab ad ba da bab bad dab dad aba ada
```

```
"C:\Program Files\Java\jdk1.8.0_101\bin\java" ...
A   Found out Diego secrets
 ab ad ba da bab bad dab dad aba ada
   Words:
ab            word
ad            word
ba            word
da            word
bab            word
bad            word
dab            word
dad            word
aba            word
ada            word
```

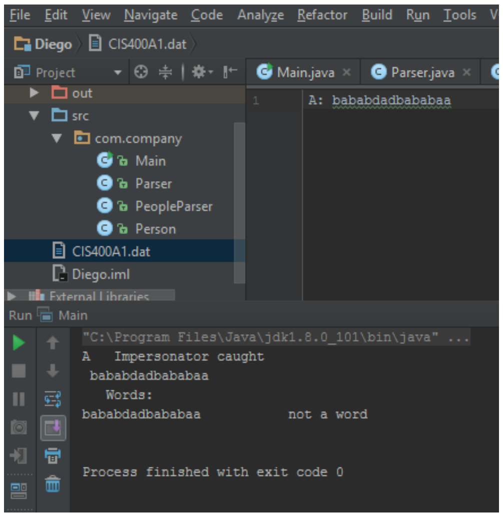
Test Case 7



```
A: babababababababa
```

```
"C:\Program Files\Java\jdk1.8.0_101\bin\java" ...
A   Found out Diego secrets
 babababababababa
   Words:
babababababababa          word


Process finished with exit code 0
```

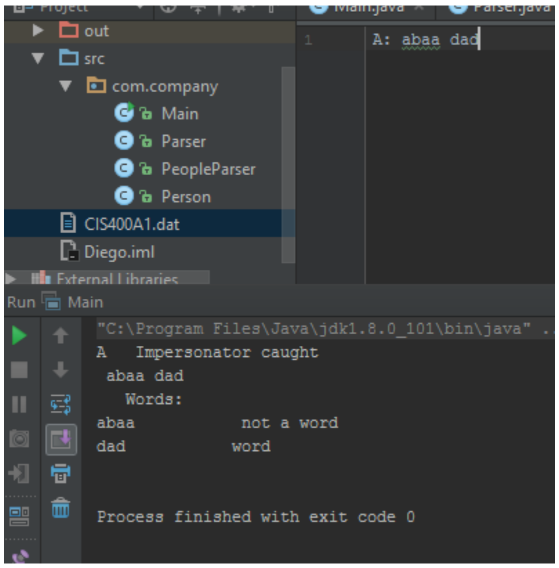
Test Case 8



```
A: bababdadbababaa
```

```
"C:\Program Files\Java\jdk1.8.0_101\bin\java" ...
A   Impersonator caught
 bababdadbababaa
   Words:
bababdadbababaa          not a word


Process finished with exit code 0
```

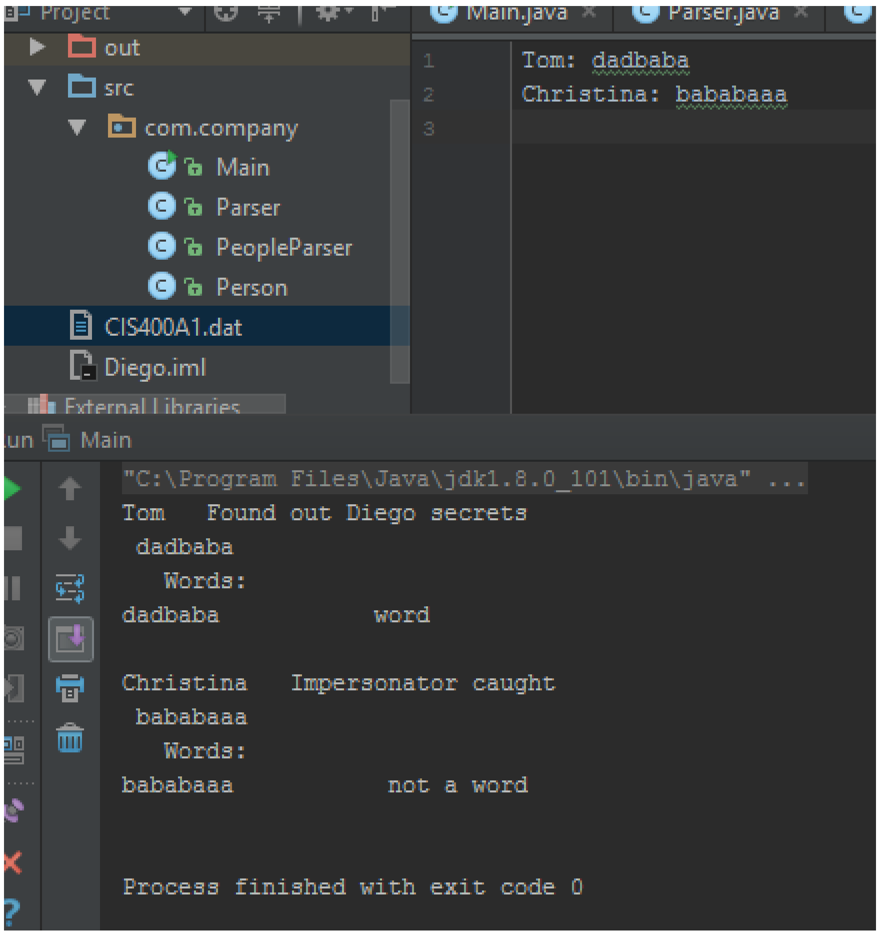
Test Case 9



```
A: abaa dad
```

```
"C:\Program Files\Java\jdk1.8.0_101\bin\java" ..
A   Impersonator caught
 abaa dad
   Words:
abaa           not a word
dad           word


Process finished with exit code 0
```

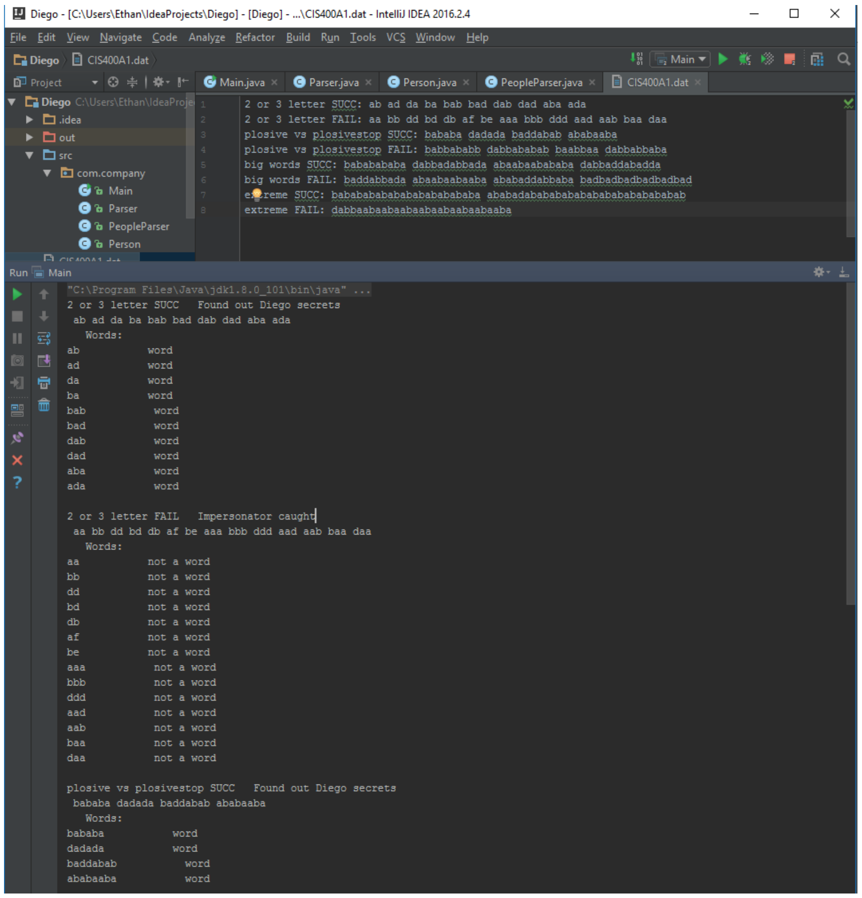
Test Case 10



```
Tom: dadbaba
Christina: bababaaa
```

```
"C:\Program Files\Java\jdk1.8.0_101\bin\java" ...
Tom   Found out Diego secrets
 dadbaba
   Words:
dadbaba           word

Christina   Impersonator caught
 bababaaa
   Words:
bababaaa           not a word


Process finished with exit code 0
```

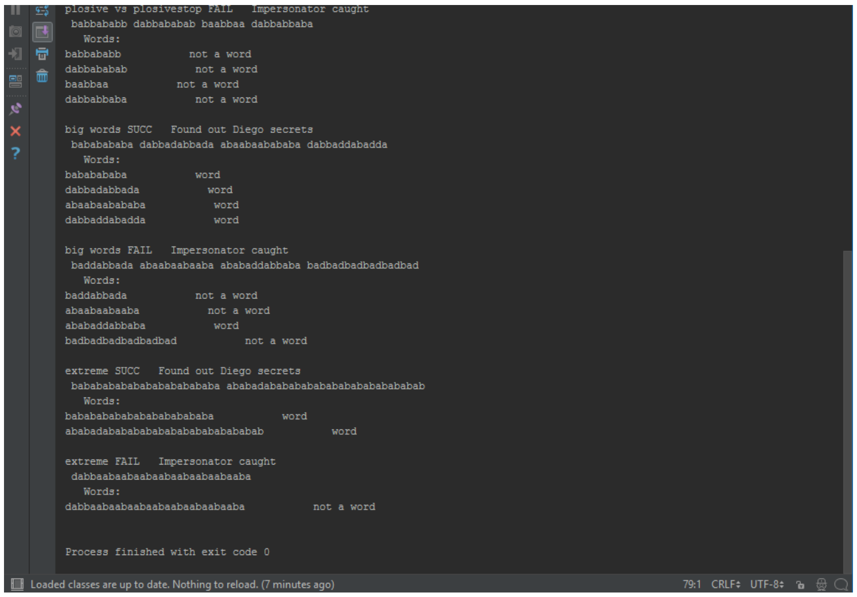
Test Case 11



Diego - [C:\Users\Ethan\IdeaProjects\Diego] - [Diego] - ...\CIS400A1.dat - IntelliJ IDEA 2016.2.4

```
2 or 3 letter SUCC: ab ad da ba bab bad dab dad aba ada
2 or 3 letter FAIL: aa bb dd bd db af be aaa bbb ddd aad aab baa daa
plosive vs plosivestop SUCC: bababa dadada baddabab ababaaba
plosive vs plosivestop FAIL: babbababb dabbababab baabbaa dabbabbaba
big words SUCC: babababababa dabbadabbada abaabaababababa dabbaddabadda
big words FAIL: baddabbada abaabaabaaba ababaddabbaba badbadbadbadbadbad
extreme SUCC: babababababababababababa ababadababababababababababababab
extreme FAIL: dabbaabaabaabaabaabaabaabaaba
```

```
"C:\Program Files\Java\jdk1.8.0_101\bin\java" ...
2 or 3 letter SUCC   Found out Diego secrets
 ab ad da ba bab bad dab dad aba ada
   Words:
ab           word
ad           word
da           word
ba           word
bab           word
bad           word
dab           word
dad           word
aba           word
ada           word

2 or 3 letter FAIL   Impersonator caught
 aa bb dd bd db af be aaa bbb ddd aad aab baa daa
   Words:
aa           not a word
bb           not a word
dd           not a word
bd           not a word
db           not a word
af           not a word
be           not a word
aaa           not a word
bbb           not a word
ddd           not a word
aad           not a word
aab           not a word
baa           not a word
daa           not a word

plosive vs plosivestop SUCC   Found out Diego secrets
 bababa dadada baddabab ababaaba
   Words:
bababa           word
dadada           word
baddabab           word
ababaaba           word
```



```
plosive vs plosivestop FAIL   Impersonator caught
 babbababb dabbababab baabbaa dabbabbaba
   Words:
babbababb           not a word
dabbababab           not a word
baabbaa           not a word
dabbabbaba           not a word

big words SUCC   Found out Diego secrets
 bababababa dabbadabbada abaabaababababa dabbaddabadda
   Words:
bababababa           word
dabbadabbada           word
abaabaababababa           word
dabbaddabadda           word

big words FAIL   Impersonator caught
 baddabbada abaabaabaaba ababaddabbaba badbadbadbadbadbad
   Words:
baddabbada           not a word
abaabaabaaba           not a word
ababaddabbaba           word
badbadbadbadbadbad           not a word

extreme SUCC   Found out Diego secrets
 babababababababababababa ababadababababababababababababab
   Words:
babababababababababababa           word
ababadababababababababababababab           word

extreme FAIL   Impersonator caught
 dabbaabaabaabaabaabaabaabaaba
   Words:
dabbaabaabaabaabaabaabaabaaba           not a word


Process finished with exit code 0
```

Loaded classes are up to date. Nothing to reload. (7 minutes ago)

12. Error Log

Error Type	Cause of Error	Solution to Error
Parsing - Logic	Cursor being set when Stop() is called, which messed up previous backtracking	Changed savedCursor to local scope
Parsing - Logic	Couldn't evaluate with <plosive><stop> because getting caught with just <plosive>	Switched order of Syllable1 and Syllable2 Created different style parsing for <word> -> <syllable><syllable><word>
Parsing – Logic	Didn't correctly parse "dadbaba" due to error	Correctly set up word -> 3-char, 2-char, word

13. Status

All test cases have been passed, and no bugs have been found when assumptions have been followed.