ASSIGNMENT 2

CPE 102 Introduction to Programming

LIM REAMSOVICHEA

HP053246

FE 12

Date: October 31, 2005

ASSIGNMENT 2

1. OBJECTIVE

The objective of this assignment is to write a Java application program for implementing a word dictionary. The dictionary can store up to 100 single English words.

2. EXPERIMENT

The class ***Dictionary*** is used to implement the command in the application program. The class diagram for the class ***Dictionary*** is given below:

Dictionary
private: wordList: String[] → array of words in the dictionary *private*: len: int → the number of words in the dictionary
public: Dictionary() → the constructor of the class Dictionary *public*: loadFromFile(): void → method to load contents of the dictionary from a file "dict.txt" *public*: findWord(wd: String): int → method to find the location of the words in the dictionary *public*: insertWord(wd: String): void → method to insert a word into the dictionary *public*: removeWord(wd: String): void → method to remove a word from the dictionary *public*: saveToFile(): void → method to save the contents of the **wordList** array into a file "dict.txt" *public*: displayDict(): void → method to display the words stored in the **wordList** array on the screen

3. REPORT

Program Structure:

In this program, there are two classes: **Dictionary** and **DictionarayApp**.

- The class **Dictionary** is used to construct the small methods: *loadFromFile, saveToFile, findWord, displayDict, insertWord* and *removeWord.*

- The class **DictionaryApp** is the main class which contains the menu, and it can call the methods in the class **Dictionary** to perform the tasks.

Program logic and suggestion:

Class **Dictionary**:

- **wordList**: is the array of the **String**s. It is declared as *private* because we do not want the users of other classes to modify it. As long as *mutator* method is not created, the **wordList** can be modified by the methods in the class **Dictionary** only.

- **len**: is the integer to denote the number of the words which have been stored in the **wordList** array. This **len** is very useful since it determines the number of the words. Again, it is declared as *private* and it can be modified in the class **Dictionary** only.

- **Dictionary()**: is the constructor of the class **Dictionary**. The constructor provides the flexible ways to manipulate the class which contains different properties or arguments. In this constructor, we define the **wordList** to be an array of size 100 and initialize **len** to be zero because when we start the dictionary, the words are not loaded yet.

- **loadFromFile()**: is the method to load the contents of the dictionary from the text file named "dict.txt".

    - The method is declared as *public, void* and *non-static* because we allow the users from other classes to be able to call this method; eventually, the method returns nothing. Besides, the user of other classes can create objects to call this method.

    - Before we are able to transfer the data from the file into the **wordList** array, we have to make sure that the file already exists. Thus, we need the "try/ catch" block to cope with it when there is an exception such as *FileNotFoundException* or *IOException* occurs.

    - To the read the data from file, there are many approaches. However, we choose the *Scanner* class to read the data line by line. *Scanner* class offers the flexible handlings rather than the combination of classes *FileReader* and *BufferedReader*.

    - *StringTokenizer* class provides the best feature to handle with the string. The useful thing of *StringTokenizer* is to identify each word of the whole line.

    - The sentinel-controlled loop "*while*" is used because we do not know in advance the number of the words in the file "dict.txt". The loop will stop when the object of the class *Scanner* meets the end of file or the number of words in the **wordList** array is over 100.

    - In this method, we take advantages of method **insertWord()**. It means that we just extract each word from the object of *StringTokenizer* and insert into the array by **insertWord()** method. This way is very helpful when we want to load many files that the words are not in lexicographic orders.

- **findWord(String wd)**: is the method to find the index location of the word provided in the **wordList** array.

    - This method is declared as *public*, *int* and *non-static* because the users from other class can access this method. Eventually, the method will return the integer and *non-static* is for object-oriented purpose.

    - The counter-controlled loop "*for*" is used because we realize the number of the words in the **wordList** by **len**. The loop is used to check whether the word provided is matched to the words in the **wordList** array. If the word is found in the **wordList** array, the method will return the integer value of the location index in the array, but it will return the value of -1 when the word provided is not found in the array.

- **insertWord(String wd)**: is the method to insert any word into the array of **wordList**.

    - The method is declared as *public*, *void,* and *non-static*.

- At first, we have to check whether the number of words in the array is full. If the number of words is over 100, we prompt the message to tell the user that the dictionary is full.

- After we know the dictionary can store the new word, we have to make sure that the new word does not exist in the array. Therefore, we exploit the benefit of method **findWord()** to check the word in the array. If the new word already exists in the array, we will prompt the message that the word already exists in the dictionary.

- The process of inserting is that we assign the new word into the last index location of the words in the array. After that we compare the new word to the word next to itself. When the words in dictionary are not in lexicographic order, the new word will swap the location with the next word until the words are in order.

- **removeWord(String wd)**: is the method to remove any word from the array.

  - The method is declared as *public*, *void*, and *non-static*.

  - This method takes advantages from the method **findWord().** To remove any word from the array, we need to know the index location of that word; hence we can use the method **findWord()** to find the index location of the word first.

  - Once we know the index location, we can move the rest of the words from that location forward by one step. However, if the **findWord()** method return the value of -1, it means that the word does not exists in the array; thus we will prompt the message that "the word does not exist".

- **saveToFile()**: is the method save the whole list of the **wordList** array into a file "dict.txt".

  - This method is declared as *public*, *void* and *non-static*.

  - We need the "try/ catch" block to prevent the unexpected errors such as *FileNotFoundException* or *IOException*.

  - We need to create the object to print into file; after that we use **close()** to save that array in buffer into hard drive.

- **displayDict()**: is the method to display all the words in the array on the screen.

  - The method is declared as *public*, *void*, and *non-static*.

  - This is the easiest method among all the methods in this class because it is just to print out all the words in the array on the screen. To do this, we use counter-controlled loop "*for*" because we know the number of words in array by **len**.

Class **DictionarayApp**:

- **main()**: is the method that application starts with.

- This method is just a shell of the application; it needs to call methods of other classes to build the complete application. Therefore, we need to create an object **dict** for class **Dictionary**.

- We use sentinel-controlled loop to control the choices of users. When the users choose an inappropriate option, the loop will ask the users to reenter.

- Since the option is the integer, *switch* is the best syntax to cope with it.

***Some suggestions to improve the application*:**

Even though this application allows users to search, insert, delete and save the words in the dictionary, there are still some weak points in logic and syntax.

- The program loads only a file, namely "dict.txt". That is not flexible because users are unable to load other files besides "dict.txt".

- In this specific program, we do not have to define the constructor of the class Dictionary since we can initialize string **wordList** of size 100 and **len** to be zero immediately.

- **findWord()** method provides the inconvenient notation because it returns the index of location of array which counts from zero, but the users may count from one.

- Sometimes, the users may forget to save the file before they quit the application; hence, the program should inform the user whether they want to save or not.

4. JAVA PROGRAMMING CODES

Class Dictionary

```
// This is a class dictionary in which we can search, insert, delete, save
// and print words.
//              Author:     Lim Reamsovichea
//              Date:       25 October 2005


import java.util.Scanner; // imported for class Scanner
import java.util.StringTokenizer; // imported for class StringTokenizer
import java.io.*; // imported for all classes in package IO


public class Dictionary
{
        private String[] wordList; // array of String wordlists used in dictionary
```

```
private int len; // the number of words in wordList array


// constructor for class Dictionary
public Dictionary()
{
	wordList = new String[100]; // wordList is declared as 100 limited words
	len = 0; // the number of words is initialized to zero

} // end of constructor



// this method will load the data/ wordlists from the file "dict.txt"
public void loadFromFile()
{

	/* the "try/ catch" block is needed since the file is called. When there
	 * is an exception, the "catch" block will catch and print out the error
	 * message.
	 */


	try
	{

		// create an object scStream to load the file "dict.txt"
		Scanner scStream = new Scanner(new File("dict.txt"));

		/* using Scanner class is more flexible than BufferedReader class */

		while(scStream.hasNext() && len < 100)
		{

			// extract each line of the file "dict.txt" into strToken
			StringTokenizer strToken = new StringTokenizer(scStream.next());

			// insert each word in the file "dict.txt" into wordList array
			insertWord(strToken.nextToken());

			System.out.print("\r");

		} /* the loop will stop when there is no more words in "dict.txt"
		   * to be assigned or the number of wordList array is more than 100
		   */


		System.out.print("\rThe file is loaded                    ");

	} // end "try"

	// error message is prompted when the file is not found
```

```
        catch(FileNotFoundException e)
        {
                System.out.println("Error: File not found " + e.getMessage());
                System.exit(0);
        }

        // error message is prompted when there is a possible exception
        catch(IOException e)
        {
                System.out.println("Error: IO error");
                e.printStackTrace(); // print the tracing exception
                System.exit(0);
        }


} // end method loadFromFile




// this method will find the word which matches to wordList
public int findWord(String wd)
{

        for(int i = 0; i < len; i++)
        {
                // it will return the index if any wordList is matching to "wd"
                if(wordList[i].equals(wd)) return i;
        }

        // it will return -1 when "wd" is not matching to any wordList
        return -1;

} // end method findWord




// this method will insert a word into dictionary
public void insertWord(String wd)
{
        // we need to convert "wd" into lower case first
        wd = wd.toLowerCase();

        if(len >= 100) System.out.println("The dictionary is full.");

        else // the number of wordList is less than 100
        {

                // to check whether the word "wd" already exists in wordList
                if(findWord(wd) != -1)
                {
                        System.out.print("The word \"" + wd + "\" already exists");
```

```
                }


                else // the word "wd" does not exist in the wordList yet
                {


                        String temp; // to exchange the locations of the wordList array
                        int i = len;

                        // the new word is assigned to the last index of wordList
                        wordList[len] = wd;


                        /* i > 0 must be written first to prevent the array out of bound
                         * when i = 0
                         */

                        while(i > 0 && wordList[i].compareTo(wordList[i - 1]) < 0)
                        {
                                // exchange wordList[i] and wordList[i-1]
                                temp = wordList[i];
                                wordList[i] = wordList[i - 1];
                                wordList[i - 1] = temp;

                                i--;
                        }

                        len++; // the number of wordList is increased by 1


                        System.out.print("The word \"" + wd + "\" is inserted");

                } // end nested "if-else"

        } // end "if-else"

} // end method insertWord




// this method will remove word from the wordList
public void removeWord(String wd)
{
        int position = findWord(wd); // find the location of the matching word

        if(position == -1) System.out.println("The word \"" + wd + "\" does not exist");

        else
        {
                // move the location of each word backward by 1 step
                for(int i = position; i < len; i++)
```

```
                {
                        wordList[i] = wordList[i + 1];
                }

                len--; // the number of wordList is decreased by 1

                System.out.println("The word \"" + wd + "\" is removed");

        } // end "if-else"

} // end method removeWord



// this method will save all the wordList array into file "dict.txt"
public  void saveToFile()
{

        /* the "try/ catch" block is needed since the file is called. When there
          * is an exception, the "catch" block will catch and print out the error
          * message.
          */

        try
        {
                // create objects to save into file
                FileWriter fwStream = new FileWriter("dict.txt");
                BufferedWriter bwStream = new BufferedWriter(fwStream);
                PrintWriter pwStream = new PrintWriter(bwStream);

                for(int i = 0; i< len; i++) pwStream.println(wordList[i]);

                pwStream.close();

                System.out.println("The file dict.txt is written");

        } // end of "try"

        // error message is prompted when the file is not found
        catch(FileNotFoundException e)
        {
                System.out.println("Error: File not found " + e.getMessage());
                System.exit(0);
        }

        // error message is prompted when there is a possible exception
        catch(IOException e)
        {
                System.out.println("Error: IO error");
                e.printStackTrace(); // print the tracing exception
                System.exit(0);
        }
```

```
	} // end method saveToFile


	// this method will display the whole list of wordList
	public void displayDict()
	{
		System.out.println("\nThe words in the dictionary:");
		System.out.println("=====================================\n");

		for(int i = 0 ; i < len; i++) System.out.println(wordList[i]);


	System.out.println("\n=====================================\n");


	} // end method display


} // end class Dictionary
```

Class DictionaryApp

```
// This is a program about a dictionaray in which we can search, insert, delete,
// save and print words.
//		Author:		Lim Reamsovichea
//		Date:		25 October 2005


import java.util.Scanner; // imported for class scanner;

public class DictionaryApp
{
	// Start of method main
	public static void main(String[] args)
	{

		// Local Definitions

		int choice;

		Dictionary dict = new Dictionary(); // create object of class Dictionary

		Scanner sc = new Scanner(System.in); // create object of class Scanner

		// Statements

		// menu of the command
		System.out.println("\n\nCommand:");
```

```
System.out.println("(1) Load the word dictionary from file");
System.out.println("(2) Save the word dictionary to file");
System.out.println("(3) Find word");
System.out.println("(4) Insert word");
System.out.println("(5) Remove word");
System.out.println("(6) Print the word dictionary");
System.out.println("(7) Quit");


do // the loop will ask user to reenter if there is an inappropriate input
{

        System.out.print("\n\nEnter the choice: "); // prompt
        choice = sc.nextInt(); // read the choice from user

        switch(choice)
        {
                case 1: dict.loadFromFile(); // load the file into wordList array
                        break;

                case 2: dict.saveToFile(); // save wordList array into file
                        break;

                case 3: System.out.print("Enter the word to find: "); // prompt
                        String findWord = sc.next(); // read the word to find

                        int index = dict.findWord(findWord);

                        if(index == -1)
                                System.out.println("The word \"" + findWord +
                                        "\" does not exist");

                        else
                                System.out.println("The word \"" + findWord +
                                        "\" is at index " + index);

                        break;

                case 4: System.out.print("Enter the word to insert: "); // prompt
                        String insertWord = sc.next(); // read the word to insert

                        dict.insertWord(insertWord); // insert a word
                        break;

                case 5: System.out.print("Enter the word to remove: "); // prompt
                        String removeWord = sc.next(); // read the word to remove

                        dict.removeWord(removeWord); // remove a word
                        break;

                case 6: dict.displayDict(); // display the wordList
                        break;
```

```
                    case 7: System.out.println("Program terminating ...");
                            break;

                    default: System.out.println("You chose the wrong option.\n");

                } // end of switch

            }while(choice != 7); // end of loop

        } // end method main

} // end class DictionarayApp
```

5. TEST CASE

```
[hp053246@c157 as2]$ java DictionaryApp

Command:
(1) Load the word dictionary from file
(2) Save the word dictionary to file
(3) Find word
(4) Insert word
(5) Remove word
(6) Print the word dictionary
(7) Quit

Enter the choice: 1
The file is loaded

Enter the choice: 6

The words in the dictionary:
====================================

compiler
data
design
information
introduction
java
logic
mathematics
mining
programming
retrieval
structure

====================================
```

```
Enter the choice: 3
Enter the word to find: information
The word "information" is at index 3

Enter the choice: 5
Enter the word to remove: information
The word "information" is removed

Enter the choice: 4
Enter the word to insert: test
The word "test" is inserted

Enter the choice: 6

The words in the dictionary:
====================================

compiler
data
design
introduction
java
logic
mathematics
mining
programming
retrieval
structure
test

====================================


Enter the choice: 3
Enter the word to find: nonexist
The word "nonexist" does not exist

Enter the choice: 5
Enter the word to remove: nonexist
The word "nonexist" does not exist

Enter the choice: 4
Enter the word to insert: retrieval
The word "retrieval" already exists

Enter the choice: 6

The words in the dictionary:
====================================

compiler
data
design
```

```
introduction
java
logic
mathematics
mining
programming
retrieval
structure
test

=====================================


Enter the choice: 2
The file dict.txt is written

Enter the choice: 7
Program terminating ...
[hp053246@c157 as2]$ exit
```


Email: singachea@pmail.ntu.edu.sg
Website: http://singachea.atspace.biz

Singachea