

CS 2210a — Data Structures and Algorithms
Assignment 4
Smart Dictionary
Due Date: November 16, 11:59 pm
Total marks: 20

1 Overview

You are to write a program which stores a set of words and their definitions in an ordered dictionary implemented using a binary search tree (you do not need to implement an AVL tree). The definition of a word can be given in different forms, it can be either

- text explaining the meaning of the word, or
- an audio file describing the word (we will consider only audio files with the extensions “.wav” or “.mid”), or
- an image file showing the meaning of the word (we will consider only image files with the extensions “.jpg” or “.gif”).

Your program must allow the user to enter the following commands. A *word* is any string of one or more letters.

- *define word*

If the specified word is in the dictionary, the program retrieves its definition, and

- if the definition is text, this text is displayed on the screen;
- if the definition is given by an audio file, that file is played;
- if the definition is given by an image file, the image is displayed on the screen.

If the word is not in the dictionary an appropriate message must be displayed.

- *delete word*

Removes the specified word from the dictionary. If the word is not in the dictionary an appropriate message is displayed.

- *list prefix*

Here *prefix* is a string of one or more letters. The program prints all the words (if any) in the dictionary that start with the specified prefix.

- *next word*

Prints the word from the dictionary that alphabetically follows the specified one (see below); if such a word does not exist because there is no word in the dictionary that lexicographically follows the given one, then an appropriate message is displayed.

- *previous word*

Prints the word from the dictionary that alphabetically precedes the specified one (see below); if such a word does not exist because there is no word in the dictionary that lexicographically precedes the given word, then an appropriate message is displayed.

- *end*

This terminates the program.

2 Classes to Implement

You are to implement four Java classes: `DictEntry`, `OrderedDictionary`, `DictionaryException`, and `Query`. You can implement more classes if you need to. **You must write all the code yourself.** You **cannot** use code from the textbook, the Internet, or any other sources: however, you may implement the algorithms discussed in class.

2.1 DictEntry

This class represents a data item in the dictionary. Each data item consists of three parts: a word, a definition, and a definition type. The type of the definition is 1 if the definition is a text string, it is 2 if the definition is given as an audio file, and it is 3 if the definition is an image file. For this class, you must implement all and only the following public methods:

- `public DictEntry(String word, String definition, int type)`: A constructor which returns a new `DictEntry` with the specified word, definition, and type. If the type is 1, then the text defining the word is stored in the `definition` `String`. If the type is 2 or 3, then `definition` stores the name of the corresponding multimedia file.
- `public String word()`: Returns the word in the `DictEntry`.
- `public String definition()`: Returns the definition in the `DictEntry`.
- `public int type()`: Returns the type in the `DictEntry`.

You can implement any other methods that you want to in this class, but they must be declared as `private` methods (i.e. not accessible to other classes).

2.2 OrderedDictionary

This class implements an ordered dictionary. You must implement the dictionary as a binary search tree. You must use a `DictEntry` object to store the data contained in each node of the tree. In your binary search tree **only the internal nodes will store information**. The leaves will not store any data. The key for an internal node will be the word from the `DictEntry` stored in that node.

When comparing two words, lexicographic or alphabetic order is to be used. So, for example, the word “algorithm” is smaller than “computer”, as “algorithm” lexicographically precedes the word “computer”. Also, “enter” is smaller than “entering”. You can use the Java method `String.compareTo(String str)` to perform lexicographic comparisons of words.

`OrderedDictionary` class must implement all and only the public methods specified in the `OrderedDictionaryADT` interface, shown below, and the constructor

```
public OrderedDictionary()
```

(you can download `OrderedDictionaryADT.java` from the course’s website).

```
public interface OrderedDictionaryADT {
    public String findWord (String word);
        /* Returns the definition of the given word, or it returns
           an empty string if the word is not in the dictionary. */
    public int findType (String word);
        /* Returns the type of the given word, or it returns
           -1 if the word is not in the dictionary. */
}
```

```

public void insert (String word, String definition, int type)
                                throws DictionaryException;
    /* Adds the word, its definition, and type into the
       dictionary. It throws a DictionaryException if the word
       is already in the dictionary. */

public void remove (String word) throws DictionaryException;
    /* Removes the entry with the given word from the dictionary.
       It throws a DictionaryException if the word is not in the
       dictionary. */

public String successor (String word);
    /* Returns the successor of the given word (the word from
       the dictionary that lexicographically follows the given
       one); it returns an empty string if the given word has
       no successor. The given word does not need to be in the
       dictionary. */

public String predecessor (String word);
    /* Returns the predecessor of the given word (the word from
       the dictionary that lexicographically precedes the given
       one); it returns an empty string if the given word has
       no predecessor. The given word does not need to be in the
       dictionary. */
}

```

To implement this interface, you need to declare your `OrderedDictionary` class as follows:

```

public class OrderedDictionary implements OrderedDictionaryADT {
    :
}

```

Note that all public methods in `OrderedDictionaryADT` must be implemented in your `OrderedDictionary` class.

You can implement any other methods that you want to in this class, but they must be declared as private methods (i.e. not accessible to other classes).

2.3 DictionaryException

This is just the class implementing the class of exceptions thrown by the `insert` and `remove` methods of `OrderedDictionary`. See the class notes on exceptions.

2.4 Query

This class contains the main method, declared with the usual method header:

```

public static void main(String[] args)

```

You can implement any other methods that you want to in this class, but they must be declared as private methods (i.e. not accessible to other classes). The input to the program will be a file containing the words and definitions that are to be stored in the ordered dictionary. Therefore, to run the program you will type this command:

```
java Query inputFile
```

where *inputFile* is the name of the file containing the input for the program. The format for this file is as follows. The first line contains a word whose definition appears in the second line. The third line contains another word with its definition in the fourth line, and so on. For example an input file might be the following one:

```
course
A series of talks or lessons, for example, CS2210.
computer
An electronic machine frequently used by Computer Science students.
homework
Very enjoyable work that students need to complete outside the classroom.
roar
roar.wav
flower
flower.jpg
```

In this example, the first 3 words are defined by text strings, while the two last ones are defined by multimedia files, `roar.wav` and `flower.jpg`. The `DictEntry` for word “course” will store “A series of talks or lessons, for example, CS2210.” as its definition and the type will be set to 1. The `DictEntry` for word “roar” will store “`roar.wav`” as the definition and the type will be set to 2; the `DictEntry` for “flower” will store “`flower.jpg`” and its type will be 3.

All words (not the definitions, just the words) will be converted to lower case before being stored in the dictionary, so that capitalization does not matter when looking for a word in the dictionary.

To simplify the testing process, you are required to use method

```
read (String label)
```

from the provided `StringReader` class to read user commands entered from the keyboard. Class `StringReader.java` can be downloaded from the course’s website. The above method prints on the screen the label supplied as parameter and then it reads one line of input from the keyboard; the line read is returned as a `String` to the invoking method. So, for example, to query the user for input you might use this in your program:

```
StringReader keyboard = new StringReader();
String line = keyboard.read(“Enter next command: ”);
```

To play a sound file you must use the provided Java class `SoundPlayer.java`, which can be downloaded from the course’s website; you will use method `play (String fileName)` to play the named audio file. To display an image, you must use the `PictureViewer.java` class, which can be downloaded from the course’s website; you will use method `show(String fileName)` to display a `.jpg` or `.gif` file.

A `MultimediaException` will be thrown by methods `play` and `show` if the named files cannot be found or if they cannot be processed. Your program must catch these exceptions and print appropriate messages.

2.5 Example

Assume that your program receives as input a file containing the 10 lines indicated above (with definitions for “computer”, “course”, “homework”, “roar”, and “flower”). A session with your program might look like this:

Enter next command:

```
define homework
```

Very enjoyable work that students need to complete outside the classroom.

Enter next command:

```
list co
```

```
computer, course
```

Enter next command:

```
next computer
```

```
course
```

Enter next command:

```
previous homework
```

```
flower
```

Enter next command:

```
delete course
```

Enter next command:

```
delete course
```

```
course is not in the dictionary
```

Enter next command:

```
next computer
```

```
flower
```

Enter next command:

```
next roar
```

```
roar is the last word in the dictionary
```

Enter next command:

```
previous computer
```

```
computer is the first word in the dictionary
```

Enter next command:

```
define roar
```

```
(roar.wav sound file is played)
```

Enter next command:

```
define flower
```

```
(file flower.jpg is displayed)
```

Enter next command:

```
end
```

Your program must print an appropriate message if an invalid command is entered.

Hint. You might find the StringTokenizer Java class useful.

3 Testing your Program

We will run a test program called TestDict to make sure that your implementation of the OrderedDictionary has the properties specified above. We will supply you with a copy of TestDict to test your implementation. We will also run other tests on your software to check whether it works properly.

4 Coding Style

Your mark will be based partly on your coding style. Among the things that we will check, are

- Variable and method names should be chosen to reflect their purpose in the program.
- Comments, indenting, and white spaces should be used to improve readability.
- No variable declarations should appear outside methods (“instance variables”) unless they contain data which is to be maintained in the object from call to call. In other words, variables which are needed only inside methods, whose values do not have to be remembered until the next method call, should be declared inside those methods.
- All variables declared outside methods (“instance variables”) should be declared `private` (not `protected`), to maximize information hiding. Any access to the variables should be done with accessor methods (like `word` and `definition` for `DictEntry`).

5 Marking

Your mark will be computed as follows.

- Program compiles, produces meaningful output: 2 marks.
- `TestDict` tests pass: 5 marks.
- Query tests pass: 3 marks
- Coding style: 2 marks.
- Ordered Dictionary implementation: 5 marks.
- Query program implementation: 3 marks.

6 Submitting Your Program

You are required to fill out and sign an Assignment Submission Form, which can be downloaded from <http://www.csd.uwo.ca/courses/CS2210a/submForm.html>. You must also print an assignment ticket. Put the form, ticket, and a hard copy of your program in a letter size envelope labelled with your name and course number. Drop this envelope by the due date in the CS2210-locker on the third floor of the Middlesex College Building.

You must also submit an electronic copy of your program. Please keep all the code for the assignment in a directory called `Assignment4`. To submit electronically your program issue the command

```
submit cs2210 Assignment4
```

Please do not put your code in sub-directories; if you do this you might be penalized as this makes it harder for the TA's to mark the assignments.

When you submit your program, we will receive a copy of it with a datestamp and timestamp. If you submit your program more than once, please send me an email message to let me know. We will take the latest program submitted as the final version, and we will deduct marks accordingly if it is late.

After you submit your assignment you should receive an email message from the submissions system acknowledging the receipt of your work. You can also check whether your assignment was received by filling out the electronic form at

```
http://www.csd.uwo.ca/undergrad/Courses/ereceipt.shtml
```

It is your responsibility to ensure that your assignment was received by the system.