CS 2210a — Data Structures and Algorithms
Assignment 4: Ordered Dictionary Implemented with Binary Search Trees

Due Date: November 18, 11:59 pm
Total marks: 20

1 Overview

For this assignment you are to write a program which stores a set of records in an ordered dictionary implemented using a binary search tree (you do not need to implement an AVL tree). Each internal node of the binary search tree will store a record of the form `(word,type,data)`, where `word`, `type`, and `data` are of type `String`. The attribute `word` of a record can be any string, while `type` and `data` can only have the following values:

- "`text`": this means that the `data` part of the record is any string;
- "`audio`": this means that the `data` part of the record is the name of an audio file. The only audio files that we will consider are of type ".wav" and ".mid".
- "`image`": this means that the `data` part of the record is the name of an image file. The only image files that we will consider are of type ".gif" and ".jpg".

1.1 Comparing Keys

In each record `(word,type,data)` the pair of attributes `(word,type)` will be used as the key. To process the binary tree we need to be able to compare the keys of the records. Comparison between two keys `(word1,type1)` and `(word2,type2)` is done in the following manner:

- the keys are equal if `word1 = word2` and `type1 = type2`;
- the key `(word1,type1)` is smaller than the key `(word2,type2)` if either:
  - `word1` lexicographically precedes `word2`, or
  - `word1 = word2` and `type1` lexicographically precedes `type2`.

根据字典上面的顺序排左右

Consider, for example, an ordered dictionary storing the following records:

- r_1 =("computer","text","An electronic machine frequently used by Computer Science students.")
- r_2 =("computer","image","computer.gif")
- r_3 =("flower","image","flower.jpg")
- r_4 =("ping","audio","ping.wav")
- r_5 =("course","text","A series of lessons, for example CS2210")

The key ("computer","text") of r_1 is larger than the key ("computer","image") of r_2 because "image" lexicographically precedes "text". Also the key of r_2 is smaller than the key ("flower","image") of r_3 because "computer" lexicographically precedes "flower".

1.2 Text-Based User Interface

Your program will provide a simple text-based user interface that will allow users to interact with the ordered dictionary through the use of the following commands.

- `get word`

  If the ordered dictionary has records containing the given `word`, then each one of these records `(word,type,data)` will be processed in the following manner:

  - if `type =` "`text`", then print `data` on the screen,
  - if `type =` "`audio`", then play the audio file whose name is stored in `data`,
  - if `type =` "`image`", then display the image stored in the file whose name is stored in `data`.

  If no record in the ordered dictionary contains the specified `word` in its key attribute then the program must print a message indicating that the word is not in the dictionary and then it must print (*i*) the word stored in the ordered dictionary that immediately precedes `word` in lexicographic order (if any), and (*ii*) the word in the ordered dictionary that immediately follows `word` in lexicographic order (if any).

  So, for example, if in the above ordered dictionary the user enters the command `get ping` your program must play the file "ping.wav". If the user enters the command `get computer`, first your program must display the image in file "computer.gif" and then it must print the text "An electronic machine frequently used by Computer Science students.". If the user enters the command `get homework` your program should print the following:

  ```
  The word homework is not in the ordered dictionary.
  Preceding word:  flower
  Following word:  ping
  ```

  If the user enters the command `get abacus` your program should print:

  ```
  The word abacus is not in the ordered dictionary.
  Preceding word:
  Following word:  computer
  ```

- `delete word type`

  Removes from the ordered dictionary the record with key `(word,type)`, or if no such record exists, it prints

  ```
  No record in the ordered dictionary has key (word,type).
  ```

- `put word type data`

  Inserts the record `(word,type,data)` into the ordered dictionary if there is no record with key `(word,type)` already there; otherwise it prints

  ```
  A record with the given key (word,type) is already in the ordered dictionary.
  ```

- `list prefix`

  Here `prefix` is a string with one or more letters. Your program must print all the words (if any) in the ordered dictionary that start with `prefix` (if `prefix` is a word in the ordered dictionary, it must be printed also). If several records in the dictionary store the same word w and w starts with `prefix`, then the word w will be printed as many times as records contain it. For example, for the above ordered dictionary if the user enters `list co`, your program must print

`computer, computer, course`

If the user enters `list ab`, your program must print

`No words in the ordered dictionary start with prefix ab`

If the user enters `list course`, your program must print

`course`

- `smallest`

  Prints the record with smallest key in the ordered dictionary. For the above ordered dictionary the command `smallest` must print: `computer,image,computer.gif`.

- `largest`

  Prints the record with largest key in the ordered dictionary. For the above ordered dictionary the command `largest` must print: `ping,audio,ping.wav`.

- `finish`

  This command terminates the program.

2 Classes to Implement

You are to implement at least these Java classes: `Pair`, `Record`, `OrderedDictionary`, `DictionaryException`, and `UserInterface`. 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 Pair 创建 对象(computer,text)

This class represents the key attribute of a record in the ordered dictionary. Each key has two parts: `word` and `type`. Attribute `word` is a string of one or more letters; the letters in `word` **must be converted to lower case**. The attribute `type` can be "`text`", "`audio`", or "`image`", as specified above. For this class you must implement all and only the following `public` methods:

- `public Pair(String word, String type)`: A constructor which initializes a new `Pair` object with the specified word and type.
- `public String getWord()`: Returns the word stored in **this** `Pair` object.
- `public String getType()`: Returns the type stored in **this** `Pair` object.
- `public int compareTo(Pair k)`: Returns 0 if the key stored in **this** `Pair` object is equal to `k`, returns -1 if the key stored in **this** `Pair` object is smaller than `k`, and it returns 1 otherwise.

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 Record 把上面的对象 跟data连接

This class represents a record in the ordered dictionary. Each record consists of two parts: a key and the data associated with the key. For this class, you must implement all and only the following `public` methods:

- `public Record(Pair key, String data)`: A constructor which initializes a new `Record` object with the specified key and data. If the type in the key is "`audio`" or "`image`", then `data` stores the name of the corresponding audio or image file.

- `public Pair getKey()`: Returns the key stored in **this** `Record` object.
- `public String getData()`: Returns the data stored in **this** `Record` object.

You can implement any other methods that you want to in this class, but they must be declared as `private` methods.

2.3 OrderedDictionary

This class implements an ordered dictionary using a binary search tree. You must use a `Record` 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 be nodes storing null `Record` objects. The key for an internal node will be the `Pair` object from the `Record` stored in that node.

Hint. You might want to implement a class `Node` to represent the nodes of the binary search tree. Each node will store a reference to a `Record` object, and references to its left child, right child, and parent. When comparing two node keys you need to use method `compareTo` from class Pair.

In the OrderedDictionary class you must implement all 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 {
   /* Returns the Record object with key k, or it returns null if such a record is
      not in the dictionary.  */
   public Record get (Pair k);

   /* Inserts r into the ordered dictionary.  It throws a DictionaryException if a
      record with the same key as r is already in the dictionary.  */
   public void put (Record r) throws DictionaryException;

   /* Removes the record with key k from the dictionary.  It throws a
      DictionaryException if the record is not in the dictionary.  */
   public void remove (Pair k) throws DictionaryException;

   /* Returns the successor of k (the record from the ordered dictionary with
      smallest key larger than k); it returns null if the given key has no
      successor.  The given key DOES NOT need to be in the dictionary.  */
   public Record successor (Pair k);

   /* Returns the predecessor of k (the record from the ordered dictionary with
      largest key smaller than k; it returns null if the given key has no
      predecessor.  The given key DOES NOT need to be in the dictionary.  */
   public Record predecessor (Pair k);

   /* Returns the record with smallest key in the ordered dictionary.  Returns null
      if the dictionary is empty.  */
   public Record smallest ();

   /* Returns the record with largest key in the ordered dictionary.  Returns null
      if the dictionary is empty.  */
   public Record largest ();
}
```

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

```
public class OrderedDictionary implements OrderedDictionaryADT {
          ⋮
}
```

You can implement any other methods that you want to in this class, but they must be declared as `private` methods.

2.4 DictionaryException

This is the class implementing the class of exceptions thrown by the `put` and `remove` methods of `OrderedDictionary`.

2.5 UserInterface

This class implements the user interface and it 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. The input to the program will be a file containing the records that are to be stored in the ordered dictionary. Therefore, to run the program you will type this command:

`java UserInterface` *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 and the data associated with it appears in the second line. The third line contains another word with its associated data 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 three words, `course`, `computer`, and `homework` have type "`text`", while `roar` has type "`audio`", and `flower` has type "`image`". The type needs to be inferred from the data. If the data only contains one string of the form "x.y", where y = ".wav" or y = ".mid" then the type is "`audio`", if y = ".jpg" or y = ".gif" then the type is "`image`"; otherwise the type is "`text`". The `Record` object for the word `"course"` will store `(("course","text"), "A series of talks or lessons, for example, CS2210.")`. The `Record` object for the word `"roar"` will store `(("roar","audio"),"roar.wav")`; and the `Record` for `"flower"` will store `(("flower","image"),"flower.jpg")`.

Important Notes

- If the main method is as above, then the name of the input file will be stored in `args[0]`.
- The `word` attribute of each key must 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 make it easier for the TA's to test your program, 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 ask the user to type some input you might use this code fragment 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.

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

Hint. You might find useful the StringTokenizer Java class and methods toLowerCase() and startsWith(String prefix) from class String.

3 Testing your Program

We will run a test program called `TestDict` to make sure that your implementation of `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 `getWord()` and `getType()` for class `Pair`).

5 Marking

Your mark will be computed as follows.

- Program compiles, produces meaningful output: 2 marks.
- `TestDict` tests pass: 4 marks.
- `UserInterface` tests pass: 4 marks
- Coding style: 2 marks.
- Ordered Dictionary implementation: 4 marks.
- `UserInterface` program implementation: 4 marks.

6 Submitting Your Program

You are required to submit an electronic copy of your program through OWL. Please do not put your code in sub-directories as this makes it harder for the TAs to mark the assignments. Also, please do not compress your files.

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.