CS 2210a — Data Structures and Algorithms
Second Programming Assignment 4

Due Date: November 20, 11:55 pm
Total marks: 20

1 Overview

For this assignment you need to implement an ordered dictionary using a binary search tree and a simple text-based user interface that allows a user to interact with the ordered dictionary.

2 Classes to Implement

You need to implement the following Java classes: `Key`, `DataItem`, `OrderedDictionary`, `DictionaryException`, and `TextInterface`. 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 the lectures.

2.1 Class Key

This class represents the key attribute of the data items that will be stored in the internal nodes of the binary search tree implementing the ordered dictionary. Each object of this class will have two `String` instance variables: `name` and `kind`.

For this class you must implement all and only the following `public` methods:

- `public Key(String word, String type)`: A constructor which initializes a new `Key` object with the specified parameters. Parameter `word` **must be converted to lower case** before being stored in instance variable `name`. Parameter `type` must be stored in instance variable `kind`.
- `public String getName()`: Returns the `String` stored in instance variable `name` of **this** `Key` object.
- `public String getKind()`: Returns the `String` stored in instance variable `kind` of **this** `Key` object.
- `public int compareTo(Key k)`: Returns 0 if **this** `Key` object is equal to `k`, returns -1 if **this** `Key` object is smaller than `k`, and it returns 1 otherwise. To compare two Key objects you need to use the following rules:
  - Two `Key` objects `A` and `B` are equal if `A.name = B.name` and `A.kind = B.kind`.
  - `Key` object `A` is smaller than `Key` object `B` if either
    - `A.name` lexicographically precedes `B.name`, or
    - `A.name = B.name` and `A.kind` lexicographically precedes `B.kind`.

  Consider for example 4 `Key` objects: `A` with attributes `name = "car"` and `kind = "image"`; `B` with attributes `name = "car"` and `kind = "image"`; `C` with attributes `name = "car"` and `kind = "sound"`; `D` with attributes `name = "house"` and `kind = "condo"`. Then `A = B`, `A < C` because "image" lexicographically precedes "sound", and `A < D` because "car" lexicographically precedes "house".

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 Class DataItem

This class represents the data items that will be stored in the internal nodes of the binary search tree implementing the ordered dictionary. Each object of this class will have two instance variables: `Key theKey` and `String content`.

For this class you must implement all and only the following `public` methods:

- `public DataItem(Key k, String data)`: A constructor which initializes a new `DataItem` object with the specified parameters.
- `public Key getKey()`: Returns `theKey`.
- `public String getContent()`: Returns `content`.

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

2.3 Class OrderedDictionary

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

(**Hints.** 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 `DataItem` object, and references to its left child, right child, and parent. You might also want to implement a class `BinarySearchTree` to represent the binary search tree implementing the ordered dictionary.)

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 OWL.

```
public interface OrderedDictionaryADT {
   /* Returns the DataItem object with key k, or it returns null if such a DataItem
      is not in the dictionary.  */
   public DataItem get (Key k);

   /* Inserts d into the ordered dictionary.  It throws a DictionaryException if a
      DataItem with the same Key as d is already in the dictionary.  */
   public void put (DataItem d) throws DictionaryException;

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

   /* Returns the successor of k (the DataItem 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 DataItem successor (Key k);
```

```
   /* Returns the predecessor of k (the DataItem 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 DataItem predecessor (Key k);

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

   /* Returns the DataItem with largest key in the ordered dictionary.  Returns
      null if the dictionary is empty.  */
   public DataItem 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 Class DictionaryException

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

2.5 Class TextInterface

This class implements the user interface and it contains the `main` method, declared with the usual method header:

```
public static void main(String[] args)
```

The input to the program will be a file containing the information that is to be stored in the ordered dictionary. Therefore, to run the program you will type this command:

`java TextInterface` *inputFile*

where *inputFile* is the name of the file containing the input for the program. In Eclipse you must configure it so reads the input file (see instructions in programming assignment 1 to see how to do this).

Each one of the data items to be stored in the ordered dictionary is a triplet `((name,kind),content)`, where `kind` can be one of the following:

- `"definition"`: in this case `content` is the definition of `name`.
- `"sound"`: in this case `content` is the name of a sound file. The only audio files that we will consider are of type ".wav" and ".mid".
- `"picture"`: in this case `content` is the name of an image file. The only image files that we will consider are of type ".gif" and ".jpg".

- `"url"`: in this case `content` is the URL of a webpage. The only webpages that we will consider have names ending in ".html".
- `"program"`: in this case `content` is the name of a program. We will only consider programs with extension ".exe".

Your `main` method will read the input file (read Section 2.5.2 to learn about the format of the input file) and it will store the corresponding DataItem objects in an ordered dictionary. Then, your `main` method will have a loop that in each iteration will ask the user to enter a command, the program will process the command as explained below, and then the process will be repeated until the user enters the command `end` that will terminate the program.

To read user commands you **must** use method

```
read (String label)
```

from the provided `StringReader` class. 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 a command you might use this code fragment in your program:

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

2.5.1 User Commands

The commands that the user can enter and which your program must process are explained below.

- `get w`

  If the ordered dictionary has DataItem objects whose Key attributes have `name = w`, then **each one of these DataItem objects will be processed in the following manner**. Let `D` be one of these DataItem objects with Key attribute `K`, where `K.name = w`, then

  - if `K.kind = "definition"`, then your program must print `D.content` on the screen,
  - if `K.kind = "sound"`, then your program must play the audio file whose name is stored in `D.content`,
  - if `K.kind = "picture"`, then your program must display the image stored in the file whose name is stored in `D.content`.
  - if `K.kind = url"`, then your program must display the content of the web page specified in `D.content`.
  - if `K.kind = "program"`, then your program must execute the program specified in `D.content`.

  If no DataItem object in the ordered dictionary contains `w` as their `name` attribute then the program must print the message:

  `"The word w is not in the ordered dictionary"`

  and then it must print (i) the `name` attribute of the DataItem object in the ordered dictionary that immediately precedes `w` in lexicographic order (if any), and (ii) the `name` attribute of the DataItem object in the ordered dictionary that immediately follows `w` in lexicographic order (if any).

  So, for example, consider an ordered dictionary storing the following DataItem objects. We denote a DataItem object as `((name,kind),content)`, where `(name,kind)` is the key attribute.

- r_1 =(("computer","definition"),"An electronic machine frequently used by Computer Science students.")
- (("computer","picture"),"computer.gif")
- (("flower","url"),"http://www.csd.uwo.ca/flower.html")
- (("ping","sound"),"ping.wav")
- (("course","program"),"course.exe")
- (("computer","program"),"compute.exe")

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`, your program must display the image in file "computer.gif", it must print the text "An electronic machine frequently used by Computer Science students.", and it must execute the program "compute.exe". For the command `get flower` your program must display the content of the webpage "http://www.csd.uwo.ca/flower.html", and for `get course` your program must execute program "course.exe".

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
```

- `remove w k`

  Removes from the ordered dictionary the DataItem object with key `(w,k)`, or if no such record exists, it prints

  ```
  No record in the ordered dictionary has key (w,k).
  ```

- `add w k c`

  Inserts a DataItem object `((w,k),c)` into the ordered dictionary if there is no record with key `(w,k)` already there; otherwise it prints

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

- `list prefix`

  Here `prefix` is a string with one or more letters. Your program must print the `name` attributes (if any) of all the DataItem objects in the ordered dictionary that start with `prefix`; if `prefix` is the `name` attribute of a DataItem object in the ordered dictionary, then `prefix` must be printed also. If several DataItem objects in the dictionary have the same `name` attribute w, and w starts with `prefix`, then the string w will be printed as many times as the number of DataItem objects in the ordered dictionary that contain it. For example, for the above ordered dictionary if the user enters `list co`, your program must print

  ```
  computer, computer, computer, course
  ```

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

  ```
  No name attributes in the ordered dictionary start with prefix ab
  ```

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

  ```
  course
  ```

- `first`

  This command must print the attributes of the DataItem object in the ordered dictionary with smallest key. For example, for the above ordered dictionary the command `first` must print: `computer,definition,An electronic machine frequently used by Computer Science students.`.

- `last`

  This command must print the attributes of the DataItem object in the ordered dictionary with largest key. For example, for the above ordered dictionary the command `last` must print: `ping,sound,ping.wav`.

- `end`

  This command terminates the program.

- If an invalid command is entered your program must print an appropriate message.

2.5.2 Format of the Input File

The format for the input file is as follows. The first line contains a string; this is the `name` attribute of the first DataItem object to store in the ordered dictionary. The second line is a string that is the `content` attribute of the first DataItem object to store in the dictionary. The third line is a string which is the `name` attribute of the second DataItem object; the fourth line contains the `content` attribute of the second DataItem object, and so on. Note that the `kind` attributes of the DataItem objects are not stored in the input file; these attributes can be obtained as explained below.

Consider the following input file:

```
homework
Very enjoyable work that students need to complete outside the classroom.
roar
roar.wav
flower
flower.jpg
computer
computerDemo.exe
course
http://www.csd.uwo.ca/Courses/CS2210a/index.html
```

In this example, the first DataItem object will have `name = "homework"`, `content = "Very enjoyable work that students need to complete outside the classroom."` and `kind = "definition"`. The second DataItem object will have `name = "roar"`, `content = "roar.wav"` and `kind = "sound"`; the third one will have `name = "flower"`, `content = "flower.jpg"` and `kind = "picture"`; the fourth one `name = "computer"`, `content = "computerDemo.exe"` and `kind = "program"`; and the last one `name = "course"`, `content = "http://www.csd.uwo.ca/Courses/CS2210a"` and `kind = "url"`.

The `kind` attributes need to be inferred from the `content` attribute. If the `content` attribute only contains one string of the form `x.y`, then

- if `y = "wav"` or `y = "mid"` then `kind = "sound"`
- if `y = "jpg"` or `y = "gif"` then `kind = "picture"`
- if `y = "exe"` then `kind = "program"`

- if `y = "html"` then `kind = "url"`

Otherwise `kind = "definition"`.

2.5.3 Important Notes and Classes Provided

- If the main method is `public static void main(String[] args)`, then the name of the input file will be stored in `args[0]`.
- The `name` attribute of each Key object must be converted to lower case before being stored in the dictionary, so that capitalization does not matter when looking for a name in the dictionary.
- To play a sound file you must use the provided Java class `SoundPlayer.java`; you will use method `play(String fileName)` to play the named sound file.
- To display an image, you must use the provided Java class `PictureViewer.java`; you will use method `show(String fileName)` to display a `.jpg` or `.gif` file.
- To execute a program you must use the provided Java class `RunCommand.java`; you will use method `run(String commandName)` to execute the specified program.
- To display a webpage you must use the provided Java class `ShowHTML.java`; you will use method `show(String url)` to render the page on the screen.
- A `MultimediaException` will be thrown by methods `play`, `run` and `show` of classes `SoundPlayer.java`, `PictureViewer.java`, `Show HTML.java` and `RunCommand.java` if the named files cannot be found or if they cannot be processed. Your program must catch these exceptions and print appropriate messages.

  We provide you with a java class called `Sample.java` that illustrates how to use methods `play`, `run` and `show` from the above classes. Study this class so you know how to use these methods.
- If you use Eclipse, to ensure that it will find all the image, sound, html, txt, and exe files, put all these files in the same directory; then in Eclipse go to Run, Run Configurations, select Arguments, on Working directory select "Other" and click on "File System" and choose the directory that contains yor files.

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

3 Testing your Program

We will run a test program called `TestDict` to check 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 instance variable should be used 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 instance variables should be declared `private`, to maximize information hiding. Any access to these variables from other classes should be done with accessor methods (like `getName()` and `getKind()` for class `Key`).

5 Marking

Your mark will be computed as follows.

- Program compiles, produces meaningful output: 2 marks.
- `TestDict` tests pass: 4 marks.
- `TextInterface` tests pass: 4 marks
- Coding style: 2 marks.
- Ordered Dictionary implementation: 4 marks.
- `TextInterface` 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**. Delete any `package` lines at the top of your java classes as using packages makes it harder for the TAs to mark the assignments. **Please do not compress your files**.

If you submit your program more than once, we will take the latest program submitted as the final version, and we will deduct marks accordingly if it is late.