CS 2210a — Data Structures and Algorithms
Assignment 4

Due Date: November 10, 11:55 pm
Total marks: 20

1   Overview

For this assignment you will implement an ordered dictionary using a binary search tree and a text-based user interface to manipulate the information stored in the ordered dictionary.
   The Java classes that you need to implement are described below. 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, chatbots, or any other sources: however, you may implement the algorithms discussed in the lectures.

2   Class Key

This class represents the key of the data items stored in the internal nodes of the binary search tree implementing the ordered dictionary. Each object of this class will have two instance variables: label and type.
   For this class you must implement all and only the following public methods:

- **public Key(String theLabel, int theType):** A constructor which initializes a new Key object with the specified parameters. Parameter theLabel must be converted to lower case before being stored in instance variable label. You can use Java method toLowerCase to do this.

- **public String getLabel():** Returns the String stored in instance variable label .

- **public int getType():** Returns the value of instance variable type.

- **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.label = B.label and A.type = B.type.

   – Key object A is smaller than Key object B if either

      * A.label lexicographically precedes B.label, or
      * A.label = B.label and A.type < B.type.

   Consider, for example, these Key objects: A with label = "car" and type = 2; B with label = "car" and type = 2; C with label = "car" and type = 3; D with label = "house" and type = 1. Then A = B, A < C, 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.

3 Class Record

This class represents the records to be stored in the internal nodes of the binary search tree. Each object of this class will have two instance variables: Key theKey and String data.

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

- public Record(Key k, String theData): A constructor which initializes a new Record object with the specified parameters.
- public Key getKey(): Returns theKey.
- public String getDataItem(): Returns data.

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

4 Class BSTNode

This class represents a node of the binary search tree. For this class you must implement all and only the following public methods:

- public BSTNode(Record item): The constructor for the class.
- public Record getRecord(): Returns the Record object stored in this node.
- public void setRecord (Record d): Stores the given record in this node.
- public BSTNode getLeftChild(): Returns the left child.
- public BSTNode getRightChild(): Return the right child.
- public BSTNode getParent(): Returns the parent.
- public void setLeftChild(BSTNode u): Sets the left child to the specified value.
- public void setRightChild(BSTNode u): Sets the right child to the specified value.
- public void setParent(BSTNode u): Sets the parent to the specified value.
- public boolean isLeaf(): Returns true if this node is a leaf; false otherwise. A node is a leaf if both of its children are null.

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

5 Class BinarySearchTree

This class represents a binary search tree. For this class you must implement all and only the following public methods:

- public BinarySearchTree(): The constructor for the class that creates a leaf node as the root of the tree.
- public BSTNode getRoot(): Returns the root node of this binary search tree.
- public BSTNode get(BSTNode r, Key k): Returns the node storing the given key; returns null if the key is not stored in the tree with root r.
- public insert (BSTNode r, Record d): Adds the record to the binary search tree with root r. Throws a DictionaryException if the tree already stores a record with the same key as d.

2

- **public void remove (BSTNode r, Key k):** Deletes the node with the given key from the tree with root r. Throws a DictionaryException if the tree does not store a record with the given key.
- **public BSTNode successor(BSTNode r, Key k):** Returns the node storing the successor of the given key in the tree with root r; returns null if the successor does not exist.
- **public BSTNode predecessor(BSTNode r, Key k):** Returns the node storing the predecessor of the given key in the tree with root r; returns null if the predecessor does not exist.
- **public BSTNode smallest(BSTNode r):** Returns the node with the smallest key in tree with root r.
- **public BSTNode largest (BSTNode r):** Returns the node with the largest key in tree with root r.

To implement any of the above methods YOU CANNOT copy the information in the binary search tree into another data structure (like an array, an array list, linked list, ...); if you do that you will receive a very large penalty.

6   Class BSTDictionary

This class implements an ordered dictionary using a binary search tree. You must use a Record object to store the data contained in each internal node of the tree. In your binary search tree only the internal nodes will store information. The leaves must be nodes storing null Record objects. The key for an internal node will be the Key object from the Record stored in that node.

In the BSTDictionary class you must implement all the public methods specified in the BSTDictionaryADT interface shown below, and the constructor:

    public BSTDictionary()

You can download BSTDictionaryADT.java from OWL.

```
public interface BSTDictionaryADT {
    /* Returns the Record with key k, or null if the Record is not in the
        dictionary.                                                         */
    public Record get (Key k);

    /* Inserts d into the ordered dictionary.  It throws a DictionaryException if a
        Record with the same Key as d is already in the dictionary. */
    public void put (Record d) 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 (Key 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 (Key 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 (Key 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 BSTDictionary class as follows:

  public class BSTDictionary implements BSTDictionaryADT

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

7   Class Interface

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 Interface *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 (check the FAQ tab in OWL if you need help with this).

Each one of the records stored in the ordered dictionary contains a key (label, type) and data, where type can be one of the following:

- 1: In this case data is a string containing a definition of label.
- 2: In this case data is a translation of label to French.
- 3: In this case data is the name of a sound file.
- 4: In this case data is the name of a music file.
- 5: In this case data is the name of a voice file (a file containing a human voice).
- 6: In this case data is the name of an image file.
- 7: In this case data is the name of an animated image file.
- 8: In this case data is the URL of a webpage.

Note. The only sound files that we will consider have the extension .wav or .mid; the only image files that we will consider have the extension .jpg or .gif; all URL's must have the extension .html.

Your main method will first read the input file as specified below and it will store the corresponding Record objects in an ordered dictionary.

7.1   Format of the Input File

The format for the input file is as follows. The first line contains a string; this is the label attribute of the first Record object to store in the ordered dictionary. The second line ℓ contains the type and data of the first Record object:

- If the first character of ℓ is '-' then the rest of the line contains the name of a sound file
- If the first character of ℓ is '+' then the rest of the line contains the name of a music file

- If the first character of ℓ is '*' then the rest of the line contains the name of a voice file.

- f the first character of ℓ is '/' then the rest of the line contains a translation to French.

- Otherwise ℓ contains a String from which your program can infer the type attribute of the record. If ℓ only contains one string of the form x.y, then

  – if y = "gif" then type = 7 and data = ℓ
  – if y = "jpg" then type = 6 and data = ℓ
  – if y = "html" then type = 8 and data = ℓ

- Otherwise type = 1 and data = ℓ.

The third line contains a label and the fourth line contains type and data; and so on.
For example, consider the following input file:

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

In this example, the first Record object will have label = "homework", data = "Very enjoyable work that students need to complete outside the classroom." and type = 1. The second Record object will have label = "roar", data = "roar.wav" and type = 3; the third one will have label = "flower", data = "flower.jpg" and type = 6; the fourth one label = "computer", data = "computer.mid" and type = 5; and the last one label = "course", data = "http://www.csd.uwo.ca/Courses/CS2210a" and type = 8.

7.2   The main Method

After reading the input file and storing the corresponding Records in the ordered dictionary, 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 exit 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: ");

7.3   User Commands

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

- define w

  If the ordered dictionary has a Record object *d* whose Key attribute has label = w and type = 1, then your program must print the data attribute of this record on the screen.

  If the above Record object is not in the ordered dictionary then the program must print the message:

  "The word w is not in the ordered dictionary"

- translate   w

  If the ordered dictionary has a Record object *d* whose Key attribute has label = w and type = 2, then your program must print the data attribute of this record on the screen. If the above Record object is not in the ordered dictionary then the program must print the message: "There is no definition for the word w"

- sound w

  If the ordered dictionary has a Record object *d* whose Key attribute has label = w and type = 3, then your program must play the audio file whose name is stored in the data attribute of this record on the screen. If the above Record object is not in the ordered dictionary then the program must print the message: "There is no sound file for w"

- play w

  If the ordered dictionary has a Record object *d* whose Key attribute has label = w and type = 4, then your program must play the audio file whose name is stored in the data attribute of this record on the screen. If the above Record object is not in the ordered dictionary then the program must print the message: "There is no music file for w"

- say w

  If the ordered dictionary has a Record object *d* whose Key attribute has label = w and type = 5, then your program must play the audio file whose name is stored in the data attribute of this record on the screen. If the above Record object is not in the ordered dictionary then the program must print the message: "There is no voice file for w"

- show w

  If the ordered dictionary has a Record object *d* whose Key attribute has label = w and type = 6, then your program must show the image file whose name is stored in the data attribute of this record on the screen. If the above Record object is not in the ordered dictionary then the program must print the message:: "There is no image file for w"

- animate w

  If the ordered dictionary has a Record object *d* whose Key attribute has label = w and type = 7, then your program must show the image file whose name is stored in the data attribute of this record on the screen. If the above Record object is not in the ordered dictionary then the program must print the message:: "There is no animated image file for w"

- browse w

  If the ordered dictionary has a Record object *d* whose Key attribute has label = w and type = 8, then your program must show the webpage whose URL is stored in the data attribute of this record on the screen. If the above Record object is not in the ordered dictionary then the program must print the message:: "There is no webpage called w"

So, for example, consider an ordered dictionary storing the following Record objects. We denote a Record object as ((label,type),data), where (label,type) is the key attribute.

- r1 =(("computer",1),"An electronic machine frequently used by Computer Science students.")
- (("computer",6),"computer.jpg")
- (("flower",8),"http://www.csd.uwo.ca/flower.html")
- (("ping",3),"ping.wav")
- (("course",5),"course.wav")
- (("computer",7),"compute.gif")

if in the above ordered dictionary the user enters the command sound ping your program must play the file "ping.wav". If the user enters the command animate computer, your program must display the animated image file "computer.gif". If the user enters the command define computer your program must print the text "An electronic machine frequently used by Computer Science students.". For the command browse flower your program must display the content of the web-page "http://www.csd.uwo.ca/flower.html".

If the user enters the command define homework your program should print the following:

The word homework is not in the ordered dictionary.

If the user enters the command animate flower your program should print:

There is no animated image file for w

- delete w k

  Removes from the ordered dictionary the Record 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 t c

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

  ```
  A record with the given key (w,t) is already in the ordered
  dictionary
  ```
- list prefix

  Here prefix is a string with one or more letters. Your program must print the label attributes (if any) of all the Record objects in the ordered dictionary that start with prefix; if prefix is the label attribute of a Record object in the ordered dictionary, then prefix must be printed also. If several Record objects in the dictionary have the same label attribute w, and w starts with prefix, then the string w will be printed as many times as the number of Record 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 label 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 all the attributes of the Record object in the ordered dictionary with smallest key. For example, for the above ordered dictionary the command first must print:
  ```
  computer,1,An   electronic   machine   frequently   used   by
  Computer Science students.
  ```

- **last**

  This command must print all the attributes of the Record object in the ordered dictionary with largest key. For example, for the above ordered dictionary the command last must print:
  ```
  ping,3,ping.wav.
  ```

- **exit**

  This command terminates the program.

- If an invalid command is entered your program must print the message
  ```
  Invalid   command.
  ```

7.4   Important Notes and Classes Provided

- Since the main method is declared as public static void main(String[] args), then the name of the input file will be stored in args[0].

- The label 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 convert a String into an integer, you can use method Integer.parseInt().

- To play a sound file you must use the provided Java class SoundPlayer.java; you will use method play(String fileName) to play the file whose name is given as parameter.

- 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 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, and Show HTML.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 your files.

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

8   Testing your Program

We will run a test program called TestDict to test your implementation of BSTDictionary. 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. You will receive some feedback from those other tests upon submitting to Gradescope.

9   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).

10   Marking

Your mark will be computed as follows.

- Program compiles, produces meaningful output: 2 marks.
- TestDict tests pass: 4 marks.
- Interface tests pass: 4 marks
- Coding style: 2 marks.
- BSTDictionary implementation: 4 marks.
- Interface program implementation: 4 marks.

11   Submitting Your Program

You are required to submit an electronic copy of your program through Gradescope. Please do not put your code in sub-directories.

Delete any package lines at the top of your java classes as using packages will not allow our autograder to run, and you will receive an output explaining why your program failed to compile.

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.

DO NOT MODIFY ANY OTHER FILES OTHER THAN THE ONES YOU HAVE TO CREATE, FAILURE TO DO SO WILL RESULT IN YOUR CODE NOT COMPILING.

BE VERY PRECISE WHEN CREATING YOUR OUTPUT: Trailing or Leading blanks, unnecessary punctuation will lead our grader to fail.