Cs2851 Lab 5 - Morse Code

Description

In this lab, you will write and test two classes that work with Morse Code: an encoder and a decoder. Here is the *Morse Code* table associating a letter or symbol with a sequence of dots and/or dashes:

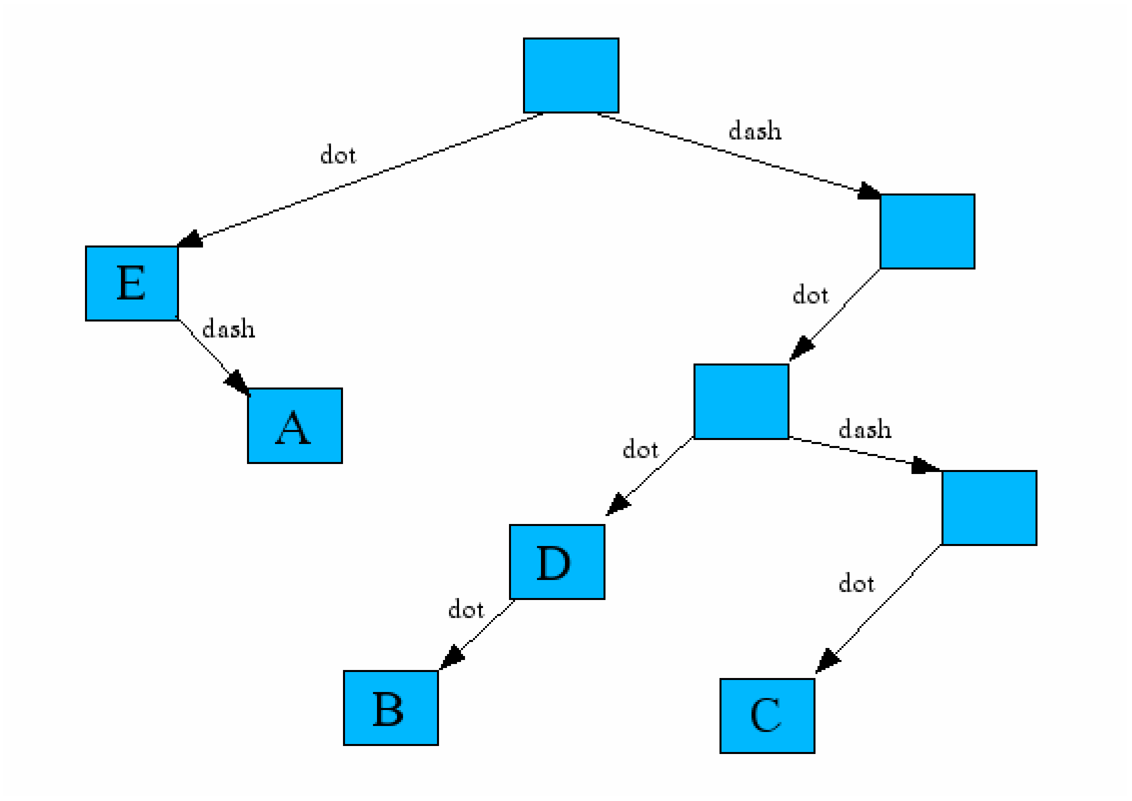
A .-	B -...	C -.-.	D -..	E .	F ..-.	G --.	H	I ..	J .---
K -.-	L .-..	M --	N -.	O ---	P .--.	Q --.-	R .-.	S ...	T -
U ..-	V ...-	W .--	X -..-	Y -.--	Z --..	0 -----	1 .----	2 ..---	3 ...--
4 ....-	5	6 -....	7 --...	8 ---..	9 ----.	Period .-.-.-	Comma --..--	Slash -..-.	Query ..--..

The class *MorseEncode* will provide a method that accepts a text string, such as SOS and returns the Morse Code equivalent. It requires a constructor that creates and populates a **TreeMap or HashMap** and one method, *encode*, that accepts a text string and returns a Morse Code string.

Here is a prototype for the *MorseEncode* class:

```
import java.util.TreeMap;

public class MorseEncode {
    private TreeMap codeMap;

    public MorseEncode() {
        . . .
    }

    /**
       *  Inserts a letter and its Morse code string into the encoding
       map.
     */
    private void addSymbol(String letter, String code) {
        . . .
    }

    /**
     *   Converts text into a Morse code message.
     *   Returns the encoded message.
     */
    public String encode(String text) {
        . . .
    }
    public static void main(String[] args) {
        MorseEncode morseEncode = new MorseEncode();

        // read input file and character-code map

        // read input
        while(s.hasNextLine()){
                String text = s.nextLine();
                if( text.length() == 0)
                        break;
                System.out.println(text + ": " +
                        morseDecode.encode( text ));
        }
    }
}
```

The *MorseDecode* class is more interesting. Here, the task is to convert a series of dots and dashes to their letter, number or symbol equivalents. To do this, build a tree using a **Tree Node** object and use it construct the complete tree as a decoder. Here is the tree, partially constructed after processing the letters *A* through *E*:



Notice that the resulting tree provides a decoding path. For example, if after becoming fully populated the tree is used to decode a *dash-dot-dash-dot* sequence, it will go *right-left-right-left* and recover the letter *C*. The boxes without letters have not been processed yet. For example, when later processing a single *dash*, the currently empty box in the upper-right corner will get the character *T*, which is corresponds to a single *dash*.

The *MorseDecode* class is structured much like the *MorseEncode* class, but it uses a user-made tree of *TreeNode* objects. The constructor creates the tree and populates it with the help of a private method, *treeInsert(char letter, String code)*. Once the tree has been constructed, the main program can call *decode(String morse)* to recover the message. Here is a prototype for the *MorseDecode* class:

```
public class MorseDecode
{
    private TreeNode decodeTree;
    private static final char DOT = '.';
    private static final char DASH = '-';

    public MorseDecode() {
        . . .
    }

    /**
     *  Inserts a letter into the decoding map based on the dot-dash
        code.
     */
    private void treeInsert(char letter, String code) {
        . . .
    }

    /**
     *   Converts Morse code message into text.
     */
    public String decode(String morse) {
        . . .
    }

    public static void main(String[] args)
    {
        MorseDecode morseDecode = new MorseDecode();

        // read input file and create decode tree

        while( (record = br.readLine()) != null) {
                String [] tokens = record.split("\\s+");
                morseDecode.treeInsert( tokens[0].trim(),
                        tokens[1].trim() );
                System.out.println(tokens[1].trim() + " " +
                        morseDecode.decode( tokens[1].trim() ));
        }
}
```

Assignment

Complete the *MorseDecode* and the *MorseEncode* classes and test them. You should use the **TreeNode** class in this project. A list of letters, numbers and symbols and their Morse Code equivalent can be found in **morsecode.txt** (tab-delimited) to construct the tree.

Requirements

1. Allow the user to enter the name of a file that contains Morse code (morsecode.txt) as a command line argument. Note: during the demo, you may be supplied with a different code file.
2. Allow user to enter English text repeatedly for encoding at the keyboard. Display the resulting Morse Code. And provide a graceful means to exit the program.
3. Allow the user to repeatedly enter the Morse code text at the keyboard for decoding at the keyboard. Display the English text. And provide a graceful means to exit the program.
4. Allow the user to display the English letters (in alphabetical order) and their Morse code equivalents.
5. Quit the program.

Lab Report

- A discussion of how you approached the problem. This should contain a sufficient level of detail
- Sample program output for both encode and decode
- A brief description of any problems you encountered and how you resolved them.
- How the lab could be improved.
- The Documented source code for your program.
- Be sure to log the time you spend on this project in the FAST system.

In your submission include your lab report and your entire project as a zip file and email to: urbain@msoe.edu. Your lab report and your zip archive should be named to include: “cs2851-lab5-yourname”.

Deadline: Lab report, code, and demonstration due by 12-noon, Monday, Finals week.

Credits: This lab was created by Jay Urbain.

The idea for the project came from Litvin's Java Methods text.