Longest prefix matching – A Trie based solution in Java

Given a dictionary of words and an input string, find the longest prefix of the string which is also a word in dictionary.

Examples:

```
Let the dictionary contains the following words:
{are, area, base, cat, cater, children, basement}

Below are some input/output examples:
--------------------------------------
Input String            Output
--------------------------------------
caterer                 cater
basemexy                base
child                   < Empty >
```

Solution
We build a Trie of all dictionary words. Once the Trie is built, traverse through it using characters of input string. If prefix matches a dictionary word, store current length and look for a longer match. Finally, return the longest match. Following is Java implementation of the above solution based.

```java
import java.util.HashMap;

// Trie Node, which stores a character and the children
class TrieNode {
    public TrieNode(char ch)  {
        value = ch;
        children = new HashMap<>();
        bIsEnd = false;
    }
    public HashMap<Character,TrieNode> getChildren() {
    public char getValue()                           {
    public void setIsEnd(boolean val)                {
    public boolean isEnd()                           {
```

```
    private char value;
    private HashMap<Character,TrieNode> children;
    private boolean bIsEnd;
}

// Implements the actual Trie
class Trie {
    // Constructor
    public Trie()   {     root = new TrieNode((char)0);

    // Method to insert a new word to Trie
    public void insert(String word)  {

        // Find length of the given word
        int length = word.length();
        TrieNode crawl = root;

        // Traverse through all characters of given word
        for( int level = 0; level < length; level++)
        {
            HashMap<Character,TrieNode> child = crawl.ge
            char ch = word.charAt(level);

            // If there is already a child for current c
            if( child.containsKey(ch))
                crawl = child.get(ch);
            else   // Else create a child
            {
                TrieNode temp = new TrieNode(ch);
                child.put( ch, temp );
                crawl = temp;
            }
        }

        // Set bIsEnd true for last character
        crawl.setIsEnd(true);
    }

    // The main method that finds out the longest string
    public String getMatchingPrefix(String input)  {
        String result = ""; // Initialize resultant stri
        int length = input.length();  // Find length of 

        // Initialize reference to traverse through Trie
        TrieNode crawl = root;

        // Iterate through all characters of input strin
        // down the Trie
        int level, prevMatch = 0;
        for( level = 0 ; level < length; level++ )
        {
            // Find current character of str
            char ch = input.charAt(level);

            // HashMap of current Trie node to traverse 
            HashMap<Character,TrieNode> child = crawl.ge

            // See if there is a Trie edge for the curre
            if( child.containsKey(ch) )
            {
               result += ch;          //Update result
               crawl = child.get(ch); //Update crawl to 

               // If this is end of a word, then update 
               if( crawl.isEnd() )
                    prevMatch = level + 1;
            }
            else  break;
        }

        // If the last processed character did not match
        // return the previously matching prefix
        if( !crawl.isEnd() )
                return result.substring(0, prevMatch);

        else return result;
    }

    private TrieNode root;
}

// Testing class
public class Test {
   public static void main(String[] args) {
        Trie dict = new Trie();
        dict.insert("are");
        dict.insert("area");
        dict.insert("base");
        dict.insert("cat");
        dict.insert("cater");
        dict.insert("basement");
        String input = "caterer";
        System.out.print(input + ":   ");
        System.out.println(dict.getMatchingPrefix(input)

        input = "basement";
        System.out.print(input + ":   ");
        System.out.println(dict.getMatchingPrefix(input)

        input = "are";
        System.out.print(input + ":   ");
        System.out.println(dict.getMatchingPrefix(input)

        input = "arex";
        System.out.print(input + ":   ");
        System.out.println(dict.getMatchingPrefix(input)

        input = "basemexz";
        System.out.print(input + ":   ");
        System.out.println(dict.getMatchingPrefix(input)

        input = "xyz";
        System.out.print(input + ":   ");
        System.out.println(dict.getMatchingPrefix(input)
    }
}
```

Output:

```
caterer:   cater
basement:   basement
are:   are
arex:   are
basemexz:   base
xyz:
```

Time Complexity: Time complexity of finding the longest prefix is O(n) where n is length of the input string. Refer this for time complexity of building the Trie.