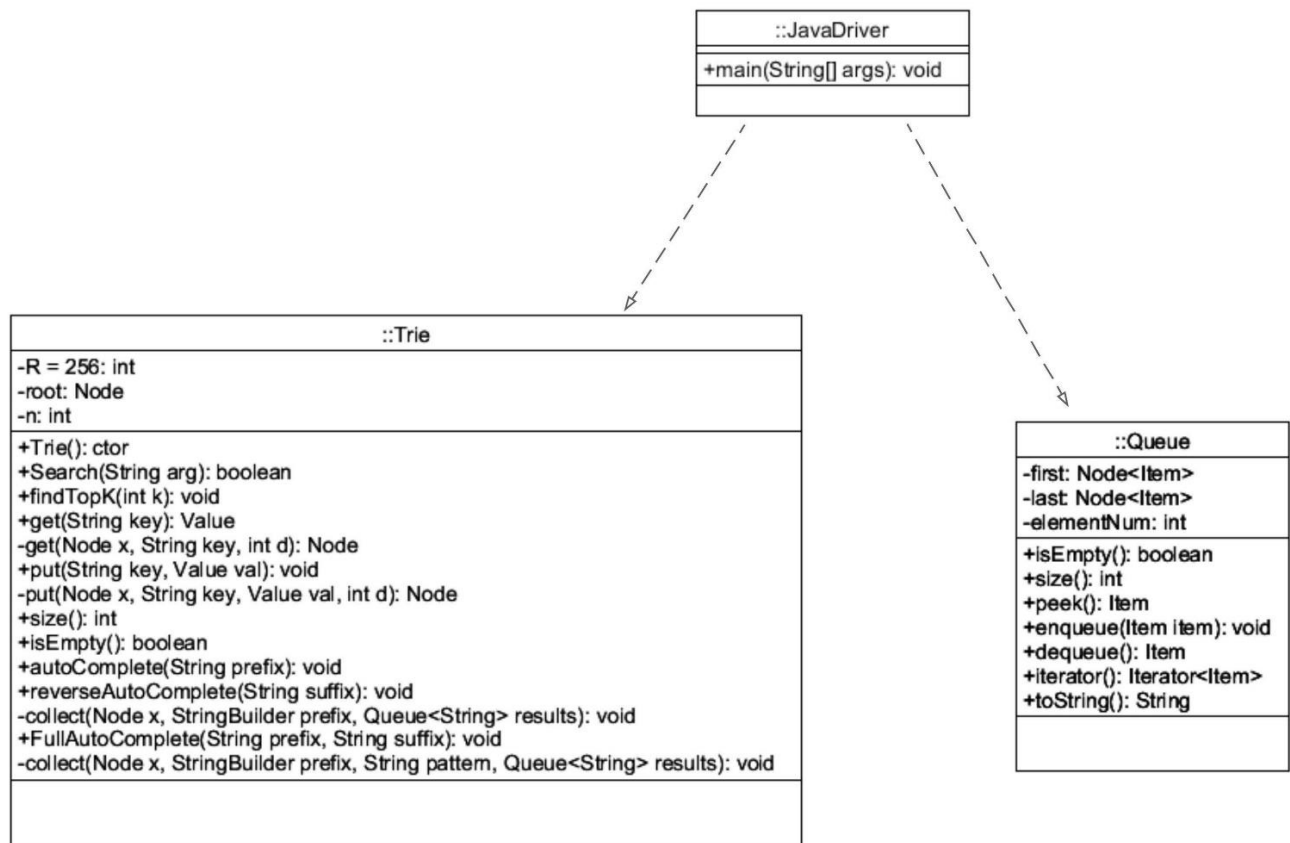


Problem Statement and Code Design:

In this homework, we are asked to implement a trie, a tree data structure to make specific operations about Strings in the trie. In total we have 3 classes which are trie, main and queue. In the trie, most of the methods are implementing with the helping of queue class. In the main method, we organized the overall structure of the homework, by creating a console application that we enter specific numbers to observe our Strings and Tries.



Implementation, Functionality, Performance Comparison:

In the first step, we created our trie class which we make Search, autoComplete (boolean Search(String arg), autoComplete (void autoComplete(String prefix)), reverse autoComplete (void reverseAutoComplete(String suffix)), FullAutoComplete (void FullAutoComplete(String prefix, String suffix)), and FindTopK (void findTopK(int k)) methods. In order to make this implementation accurately, we also created a queue class, which we use it under some methods. In the driver, those methods are used and some of the String structures are used to get a desired output.

- Trie Class

boolean Search(String arg): Checks the string key which is arg. Returns true if key is not null.

void findTopK(int k): Prints the top k words that have most occurrences.

Value get (String key): Returns the value associated with the key in the trie. If it is not in the trie, returns null.

Node get(Node x, String key, int d): Private helper method. Returns the desired value if it is in the trie. If the node is null, returns null.

void put(String key, Value val): Inserts a new key-value pair to the trie.

Node put(Node x, String key, Value val, int d): Private helper method. Makes insert operation.

int size() : Finds the size of trie.

boolean isEmpty(): Returns false if the trie is empty.

void autoComplete(String prefix): Prints all strings starts with given prefix in the trie.

void reverseAutoComplete(String suffix): print all strings end with given suffix in the trie.

void collect(Node x, StringBuilder prefix, Queue<String> results): If the value is null, inserts the prefix that is given in the parameter to the queue.

void FullAutoComplete(String prefix, String suffix): prints all strings start with given prefix and end with given suffix in the trie.

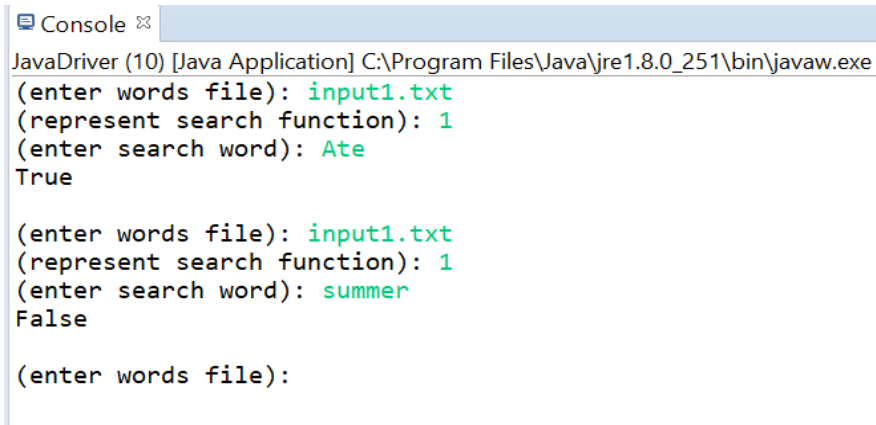
void collect(Node x, StringBuilder prefix, String pattern, Queue<String> results): Private helper method.

Time and Space Complexities of Trie

The space complexity of building a Trie is $O(n*m*\text{size of character set})$ where n is a number of keys in the trie, and m is a length of the key. For time complexity, $O(n*m)$, however in the base case it can be integrated into constant time operation which is $O(1)$.

Testing

For testing, in the main method we have a file object in order to get an input from text file and write it on to the array. We created while and if-else condition to represent the choices of user, which are numbers. Each if-else cases related with the specific methods in the trie class. For instance, after pressing 1, user can be able to find a specific word in the Trie. We completed this part by implementing a search method in the Trie class. Accordingly, we print True or False. Additionally, we used similar approaches in the other parts for our implementation by benefiting Trie class structure. Input/Output examples are as follows:

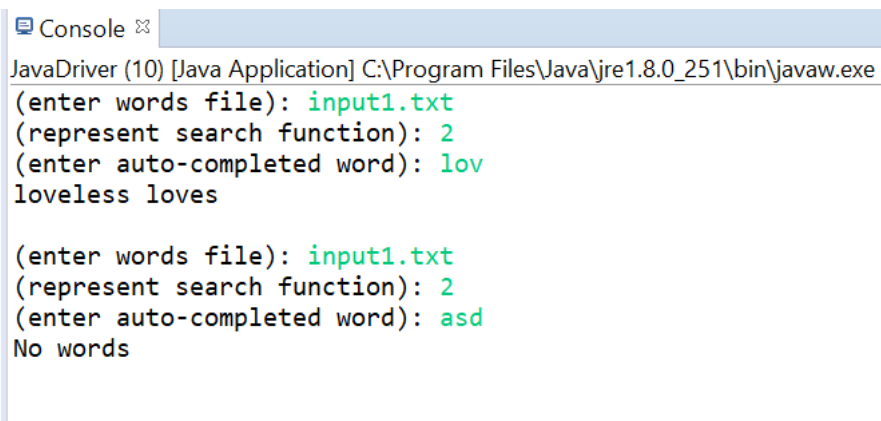


```
Console
JavaDriver (10) [Java Application] C:\Program Files\Java\jre1.8.0_251\bin\javaw.exe
(enter words file): input1.txt
(represent search function): 1
(enter search word): Ate
True

(enter words file): input1.txt
(represent search function): 1
(enter search word): summer
False

(enter words file):
```

Figure-1: Check if the word is in the text or not.



```
Console
JavaDriver (10) [Java Application] C:\Program Files\Java\jre1.8.0_251\bin\javaw.exe
(enter words file): input1.txt
(represent search function): 2
(enter auto-completed word): lov
loveless loves

(enter words file): input1.txt
(represent search function): 2
(enter auto-completed word): asd
No words
```

Figure-2: Check the words starting with given prefix.

```
Console
JavaDriver (10) [Java Application] C:\Program Files\Java\jre1.8.0_251\bin\javaw.exe
(enter words file): input1.txt
(represent search function): 3
(enter reverse auto-completed word): s
us, loves, tacos, was, is, loveless,
```

Figure-3: Check the words ending with given suffix.

```
Console
JavaDriver (10) [Java Application] C:\Program Files\Java\jre1.8.0_251\bin\javaw.exe
(enter words file): input1.txt
(represent search function): 4
(enter full auto-completed word): th ee
three

(enter words file): input1.txt
(represent search function): 4
(enter full auto-completed word): thc ee
No word
```

Figure-4: Check the words starting with given prefix and ending with given suffix.

```
Console
JavaDriver (10) [Java Application] C:\Program Files\Java\jre1.8.0_251\bin\javaw.exe
(enter words file): input1.txt
(represent search function): 5
(enter number K): 2
a, we
```

Figure-5: Top k words that have most occurrences.