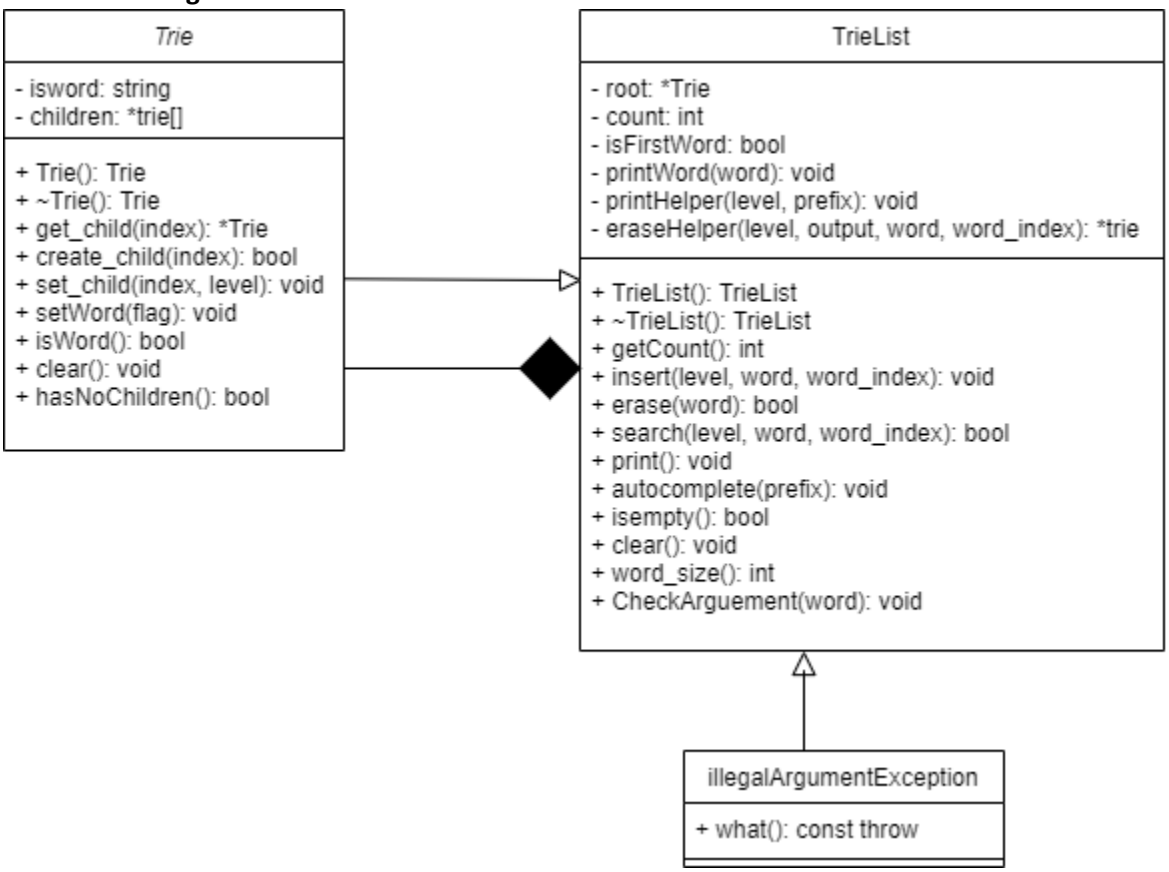
1. **Overview of Classes**

   There are three classes implemented in my project design: Trie, TrieList and illegalArgumentException.

   - Trie class stores the individual node and contains a Boolean that is used to determine if the node is the end of a word and also stores an array of pointers of type trie that point the current trie's children. The size of the array is set to 26, one index per letter of the alphabet.
   - The Trielist class is used as a wrapper class for the Trie class and stores the root of the list and the size.
   - illegalArgumentException class is used to throw error statement if operations that are invoked on the trieList contain illegal characters.

2. **UML Class Design**



3. **Details on Design Decisions**

   Trie Class:

   - Trie stores boolean value to determine if the current Trie level is a word and stores an array of pointers to the children.
   - Invoked Trie as class to make attributes private and created getter and setter functions to make changes to the Trie outside of the Trie class.

   TrieList Class:

   - Stores root, size of the tree and function helpers specific to TrieList class as private member variables.

- Constructor initializes root, sets children to NULL and sets count to zero.
- Get_Count returns the size of the TrieList.
- Insert adds new word to trie by recursively invoking Trie set_child function to insert the entire word, also checks to make sure word contains valid characters and determines corresponding print message depending on if word is in the TrieLiest.
- Erase removes a word from the trie by invoking the erase helper function. Within the erase helper function, we recursively call erase on the child and will either keep it or delete depending on if the node is an end of word or contains children.
- Search determines if input word is found within the trie, by first checking if the word contains valid character and recursively calling search on the next node to determine if it exists.
- Print, uses print helper function to print word by storing word in prefix and looping through all children and recursively calling the print function to update the prefix until we reach end of word then print word to the console.
- Clear, invokes calls clear function from Trie class that recursively removes all children at current node.
- CheckArgument loops through word to determine if it contains invalid characters, if it does then exception from illegalArgumentException class is thrown.
- isEmpty loops through all children at the root to determine if TrieList is empty
- Autocomplete moves to last letter of prefix and calls print helper function to print all words branched off the prefix.

4. Test Cases

- Added additional test cases to implement basic functionality and specifications of the project
- Tested string manipulation to ensure correct words and prefixes were extracted correctly
- Tested logical handling of running commands if TrieList is empty
- All words are removed after calling clear
- Words are properly removed after calling erase
- Logical handling with exception class if word contained invalid characters

5. Performance Considerations

To ensure run time requirements are met, I implemented the insert, erase, search, print, autocomplete and clear functions using recursion by calling the function on the current Trie's child and use the word length or the number of characters in the TrieList to indicate the number of recursive calls. Therefore, the run time for insert, erase and search are all based on the length of the word, O(n). The run time for the remaining functions, clear, autocomplete, and print depend on the number of characters already in the trie, which can be evaluated as O(N). Some other functions also use for loops however, this is still a constant run time because we are looping through the array index of size 26, this value is constant because it is based on the size of the of valid characters and not the length of the word or number of words already in the TrieList.