CS 635 Advanced Object-Oriented Design & Programming
Fall Semester, 2022
Doc 10 Assignment 1
Sep 29, 2022

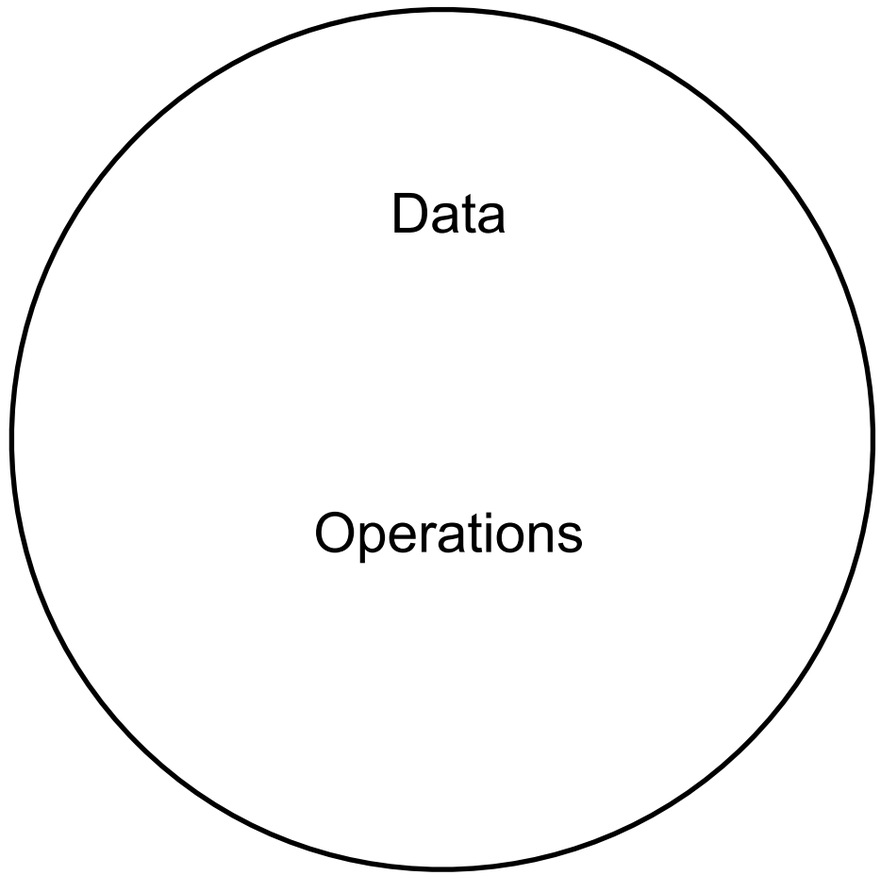
```
class Node {
    public Node[ ] children = new Node[ALPHABET_LENGHT];
    public boolean endOfTrie;

    Node() {
        this.endOfTrie = false;
    }
}
```

Struct -6

Data but no operations

```
class Node {
    public Node[ ] children = new Node[ALPHABET_LENGHT];
    public boolean endOfTrie = false;
}
```

```
class Node {
    public Node[ ] children = new Node[ALPHABET_LENGHT];
    public boolean endOfWord = false;
}
```


Data
Operations

```
class Node {
    private Node[ ] children = new Node[ALPHABET_LENGHT];
    private boolean endOfTrie = false;

    public void setEndOfTrie(Boolean value) {
        endOfTrie = value;
    }

    public Boolean isEndOfTrie() {
        return endOfTrie;
     }

    public getChildern() {
        return children;
    }

}
```

Still Struct -6

Data but no operations

Heuristics

Beware of classes that have many accessor methods defined in their public interface

Do not create god classes/objects in your system

Beware of classes that have too much noncommunicating behavior

Util Method

```
private Node search(Node node,String prefix) {
    if (reaffix.length() == 0) {
        return node;
    }
    Node next = node.getNode(prefix.charAt(0));
    if (next == null)
        return null;
    }
```

Utility method -6

Method only uses arguments

Does not use any fields

So can be anywhere - just a function

```
public class Trie {
    private static Node root;

    etc
```

```
Trie a = new Trie();
Trie b = new Trie();
a.add("Cat");
assertTrue(b.contains("Cat");
```

```
public class Trie {
    private Node root;

    public Node getRoot() {
        return root;
    }
```

Lack of information hiding -6

```
public class Trie {

    public void insert(String word) {
        etc
    }
```

```
    public boolean add(String word) {
        etc
    }
```

```
public class Trie {

    public List<String> getByFilter(String filter) {
        etc
    }

    public List<String> filter(String filter) {
        etc
    }
```

```
public class Constants {
    private static final int UpperBound = 123;
    private static final int LowerBound = 96;
    private static final int Diff = 32;

    public static int getUpperBound() {
        return UpperBound;
    }

public static int getLowerBound() {
    return LowerBound;
}

public static int getDiff() {
    return Diff;
}
```

```
public class Constants {
    public static final int UpperBound = 123;
    public static final int LowerBound = 96;
    public static final int Diff = 32;
}
```

```
public class Constants {
    public static final int UPPER_BOUND = 123;
    public static final int LOWER_BOUND = 96;
    public static final int DIFF = 32;
}
```

Contants

Poor name - constants for what?

DIFF

What is this

II

```
public class Trie {

    public void addString(String word) {
        try {
            word = verifyInput(word);
        } catch(Exception e) {
            return;
        }
        etc
    }
```

What does verifyInput do?

```
private String verifyInput(String inputString) {
    inputString = inputString.toLowerCase().replaceAll(" ","");

    if (inputString.length() > CHAR_ARRAY_SIZE || inputString.isEmpty()) {
        System.out.println("Trie can not accept \"" + inputString + "\"");
        throw new Exception("Invalid String");
    }
    return inputString;
```

```
public class Trie {

    public void allWords() {
        etc

            System.out.println(word);
        etc
    }
```

```
class Trie(object):
    find_word(node, word, substring):
```

Information hiding

Client code does not want know about node

Can not change structure of Trie without changing all client code

```
class Trie(object):
    find_word(self, node, word, substring):
```

```
class Trie(object):
    def __init__(self):
        """
        Constructs all necessary attributes for the node object
        """
        self.root = TrieNode("")
```

Duh

It is the constructor so it should construct all needed attributes for the **trie**

class TrieOperations:

Just the operations? No data?

Classes are things not operations

Names should be nouns

```
class TrieOperations:
    def print_all(Self, print_words = True):

        bunch of code

        return result_words
```

Name lies - does not print anything

```
class Trie:
    def find_word_in_trie(self, word):
```

```
class Trie:
    def find_word(self, word):
```

```
class Trie:
    def contains(self, word):
```

```
class Trie:
    def keys(self, substring: str = None) -> list:

        What is keys?
```

```
class TrieNode:

    def __init__(self, char):
        # The value of each node is a single character
        self.value = value

        # Whether or not this node has children
        self.isTerminal = False

        # Child nodes where key is a character
        self.children = {}
```

Pep 8

Function and Variable Names

Function names should be **lowercase**, with words **separated by underscores** as necessary to improve readability.

Variable names follow the same convention as function names.

mixedCase is allowed only in contexts where that's already the prevailing style (e.g. threading.py), to retain backwards compatibility.

Method Names and Instance Variables

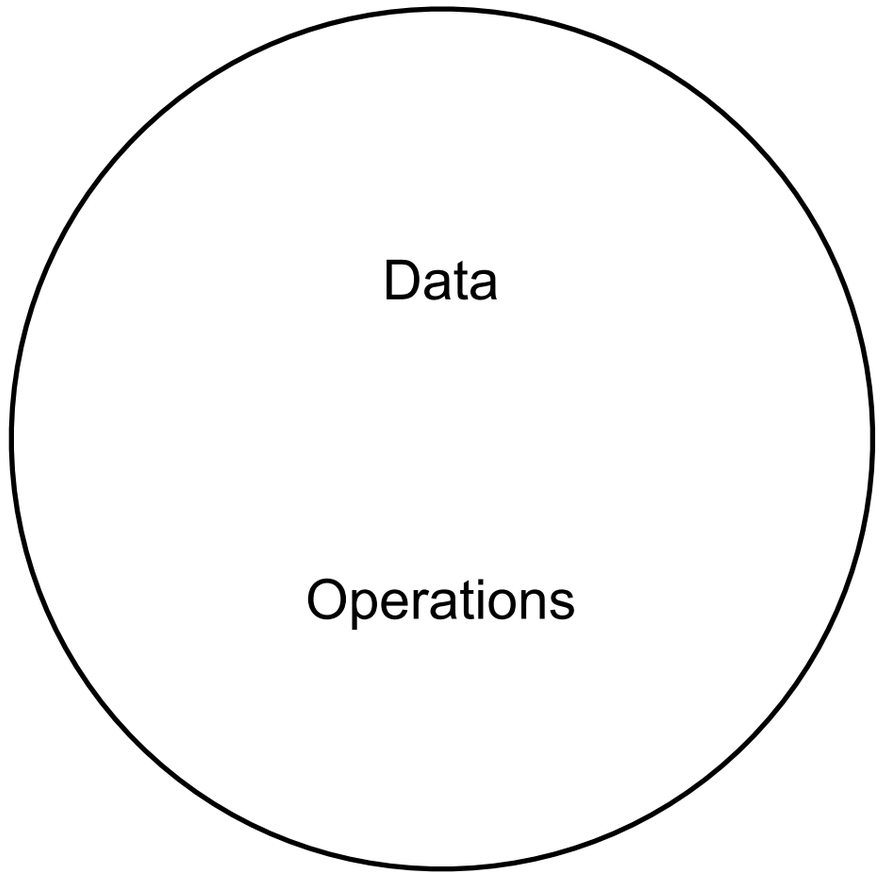
Use the function naming rules: **lowercase** with words **separated by underscores** as necessary improve readability.

Use one leading underscore only for non-public methods and instance variables.

```
class TrieNode:

    def __init__(self, char):
        self.value = value
        self.end_of_word = False
        self.children = {}
```

Struct -6

Missing operations



```
class TrieNode:
    """A node in the trie structure"""

    def __init__(self, char):
        # the character stored in this node
        self.char = char

        # mark if this node is the end of a word
        self.is_end = False

        # a dictionary for child nodes
        # (keys are characters, values are nodes)
        self.children = {}
```

```
class TrieNode:
    """A node in the trie structure"""

    def __init__(self, char):
        # the character stored in this node
        self.char = char

        # whether this can be the end of a word
        self.is_end = False

        # a counter indicating how many times a word is inserted
        # (if this node's is_end is True)
        self.counter = 0

        # a dictionary of child nodes
        # keys are characters, values are nodes
        self.children = {}
```

https://albertauyeung.github.io/2020/06/15/python-trie.html/

```
class Trie:
    def insert(self, word):
        word = word.lower();
        curr = self.root:
        for char in word:
            curr = curr.nodes[char]
        curr.is_end = True
        print("Successfully inserted", word, "in the tie")
```

Who are you talking to?

```
class State():
    # definition of class that is going to be used to keep track of the site of our trie
    def __init__(self, current_node, pattern, patternidx, words=set(), acc=[ ],
                       exclude=set(), parent=Node):
        self.curent_node = current_node
        self.pattern = pattern
        self.patternidx = patternidx
        self.acc = acc
        self.exclude = exclude
        self.parent = parent
        self.words = words
```

Struct - just data

What is this going to be used for?