CS 635 Advanced Object-Oriented Design & Programming
Fall Semester, 2022
Doc 18 Assignment 2
Nov 15, 2022

```
class Binary_Search_Tree: Name -1
    def __init__(self, strat : Strategy):
        self.orderer = strat

    def insert(self, root, new_node): -4 Info Hiding
        root.insert(new_node, self.orderer)



BST = Binary_Search_Tree(Red_ID_Strategy())
root_node = Node('foo', 'bar', '88888888', 3.4)
BST.insert(root_node, Node('foo', 'foo', '1111111', 4.0))
```

```
class BinarySearchTree:
    def __init__(self, strategy : Strategy):
        self.orderer = strategy

    def insert(self, element):
        new_node = Node(element)
        if (__root == nil):
            __root = new_node
        else:
            __root.insert(new_node, self.orderer)
```

```
BST = Binary_Search_Tree(Red_ID_Strategy())
foo = Student('foo', 'bar', '88888888', 3.4)
BST.insert(foo)
```

Name Structure - Language Conventions

	Java	Smalltalk	C#	Ruby
Class	PascalCase	PascalCase	PascalCase	PascalCase
Method	camelCase	camelCase	PascalCase	foo_bar
Field	camelCase	camelCase	camelCase	@foo_bar
Parameter	camelCase	camelCase	camelCase	foo_bar
Local Variable	camelCase	camelCase	camelCase	foo_bar

PascalCase ArrayList

camelCase courseSize

Grading Policy - Names

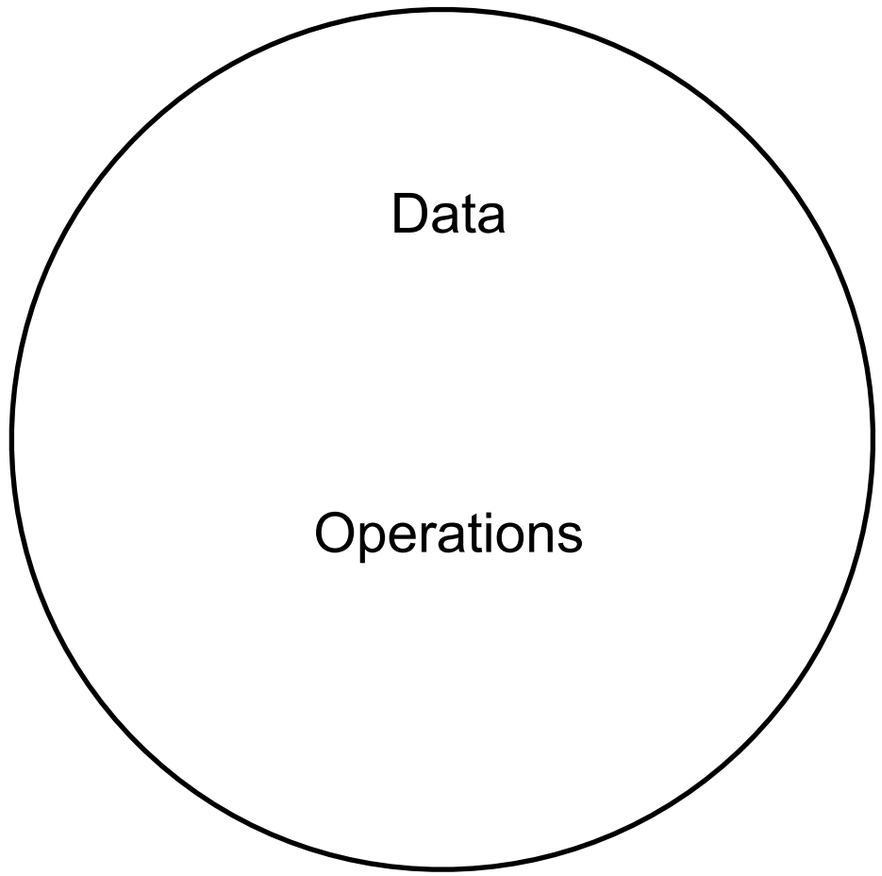
Each name that does not following your languages naming structure

- Loses 1 point/name
- Up to 20 points/assignment

Remedial Points

Each assignment I indicate points for different things

Remedial points (names, formatting) are in addition to those


Data
Operations

```
System.out.println("Enter an order");
stringOrder = readInput();
order = convertStringToOrder(stringOrder);
bst = new BST(order);

System.out.println("Enter a first name");
firstName = readInput();
System.out.println("Enter a last name");
lastName = readInput();
System.out.println("Enter a Red Id");
redId = readInput();
System.out.println("Enter a GPA);
gpa = readInput();
studentA = new Student(firstName, lastName, redId, gpa);
BST.add(studentA);

//repeat the above for studentB

println(bst.toString();
```

```
student1 = new Student("A", "B", "123456789", 3.9);
student2 = new Student("C","D", "987654321",2.6);
bst = new BST(new GPAOrder());
bst.add(student1);
bst.add(student2);
System.out.println(bst.toString());
```

```
public class Student {
    String name;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
```

```
public class TestStudent {
    @Test
    void testName() {
        Student jose = new Student();
        jose.setName("Jose");
        assertEquals("Jose", jose.getName());
    }
```

Test everything that can break

Utility Method

```
public class InnerNode {
    private Student data;
    private TreeNode left;
    private TreeNode right;

    public TreeNode add(TreeNode current, IBSTSort sortingType, Student student) { -2 Utility functio
        int sortingPriority = sortingType.compareNodes(this,student);
        if(current instanceof LeafNode){
            current = new InnerNode(student);
            return current;
        }
        else if(sortingPriority > 0){
            ((InnerNode) current).left = add(current.getLeft(),sortingType,student);
        }
        else if (sortingPriority < 0){
            ((InnerNode) current).right = add(current.getRight(),sortingType,student);
        }
        return current;
    }
```

public class Student extends Node

Student is not a type of node

A is a type of B => A is subclass of B

A has an object of type B => A has a field of type B

```
public class BSTNode {
    Student value;
    BSTNode left;
    BSTNode right;
```

Tree only useable in Assignment

```
public class BSTNode<T> {
    T value;
    BSTNode left;
    BSTNode right;
```

Can use tree in other places

```
public class BST<T> {
    private Node<T> root;

    public Node<T> getRoot() {
        return root;
    }
```

No information hiding

```
public class BST<T> {
    private Node<T> root;

    public forEach(Function<Node, Node> lambda, Node  currentNode) {
```

Names

```
public class BST<T> {
    private Node<T> root;

    public for_each(Function<Node, Node> lambda, Node  current_node) {
```

```
public class BinarySearchTree {
    private Node root;
    public NullNodeCounterVisitor null_node_visitor;
    public PathVisitor path_visitor;
```

- Visitors should not be part of Tree state
- When create a new visitor type need to edit the tree

```
public class BST<T> {


    public boolean add(T item) {
        blah
        blah
        if (strategy instance of GPAStrategy {
            more blah
        } else if (strategy instance NameStrategy {
```

Not the strategy pattern!

To add more strategies need to edit BST class

```
public class BST {
    private Node root;
    private TreeOrder order;

    public BST(int orderType) throws Exception {
        if (orderType > 2 || orderType < 0) {
            throw new Exception();
        }

        if (orderType == 0) {
            order = new RedIdOrder();
        }

        if (orderType == 1) {
            order = new LastNameOrder();
        }
```

If create a new order have to edit the tree

```
public class BST {

    public void iterator() {
        this.forEach(node -> {System.out.println(node.studentObj.firstName);}
    }
```

```
root = None
insert(root, Student(1111,"Foo", "bar", 3.9)
insert(root, Student(2345."Cat", "Dog". 2.7)

print(root.inorder_traversal())

print(sortByName(root))
```

insert, sortByName are functions

sortByName does not follow Python naming conventions

inorder_traversal does

printing makes for poor tests

```
class BinarySearchTree:
    def __init__(self, sortingMethodName):
        match sortingMethodName:
            case "RedID":
                self.sortingMethod = lambda student: student.redID
            case "Name":
                self.sortingMethod = lambda student: student.lastname + " " + student.firstName
            case default:
                raise self.invalidSortingMethodName()
        self.root = self.nullNode()

    class invalidSortingMethodName(Exception):
        def __init__(self):
            super.__init__("Please enter 'RedID' or 'Name'")

    def iterate(self, modifier):
        sel.root.iterate(modifier)
```

```
def searchBST(root, red_id):
    if root.isNone():
        return ("No student with redId- '{0}'".format(red_id)
    root_node = root
    while not root_node.isNone():
```

```
class Visiter:
    @abstractmethod
    def visit(self, node):
        pass
```

```
class Student(Node):
    def __init__(self, first_name str, last_name str, red_id int, gpa: float):
        self._first_name = first_name
        self._last_name = last_name
        self._red_id = red_id
        self._gpa = gpa
        self._left_child = NullNode()
        self._right_child = NullNode()
```

Mystery Parameters

```
class BST:
    def insert(self, root, node_values, prim_pairs, sec_pairs):
    """
    Avoid Null checks inside the BST class during insert
    """


    return AbstractNode().insert(root, node_values, prim_pairs, sec_pairs)
```

```
class AbstractNode(ABC):
    def insert(self, root, node_values, prim_pairs, sec_pairs):
    """
    Calling appropriate class's (NullNode or Node) insert method
    """


    if node:
        return Node().insert(root, node_values, prim_pairs, sec_pairs)
    return NullNode().insert(root, node_values, prim_pairs, sec_pairs)
```

Mystery Parameters

```
class Node(AbstractNode):

    def insert(self, root, node_values, prim_pairs, sec_pairs):
    """
    prim_pairs and sec_pairs have (key, value) tuple based on the Primary
    and secondary sort order strategies
    0 index will be key and 1 index will be value
    """

    if prim_pairs[1] < node.nodes[prim_pairs[0]]:
        node.left = AbstractNode().insert(root.right, node_values, prim_pairs, sec_pairs)
```

Mystery Parameters

```
def test_sort_by_redid(self):
    sort_by_redid_context = BST()
    sort_by_redid_context.set_strategy(SortByRedId())

    for node in self.input_nodes:
        self.root = sort_by_redid_context.insert_by_strategy(self.root, node)
```