CS 635 Advanced Object-Oriented Design & Programming
Fall Semester, 2021
Doc 6 Assignment 1
Sep 14, 2021

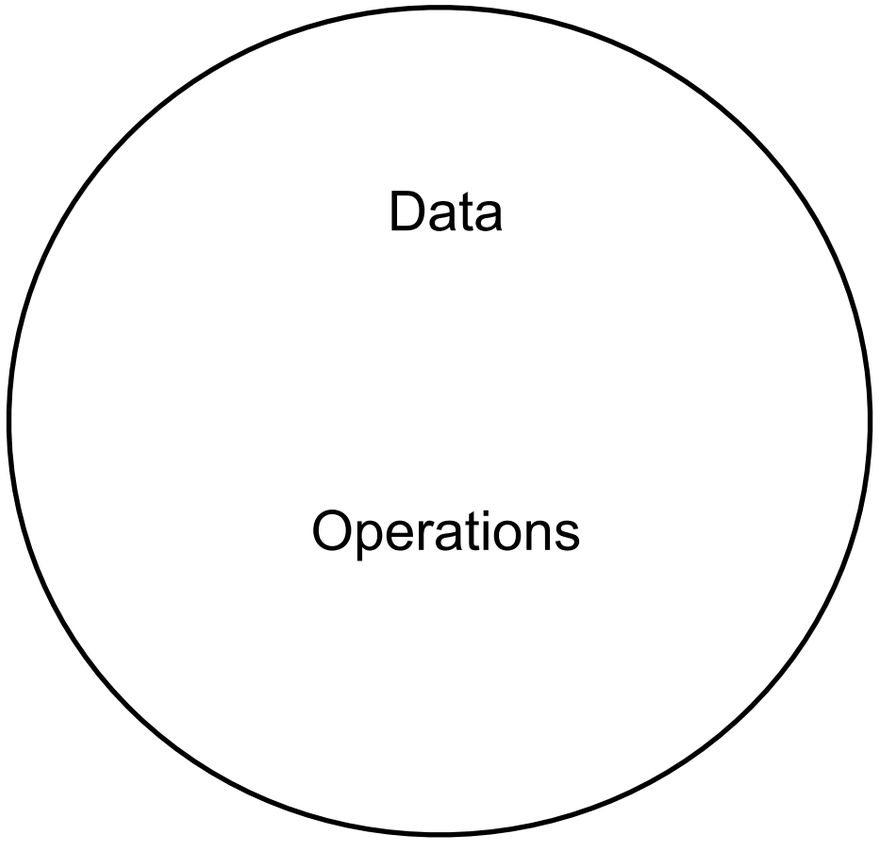
Don't Panic

Assignment 1 counts 1% of your Grade

Keep related data and behavior in one place

A class should capture one and only one key abstraction

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



If Don't Keep Data & Operations together Get

Class with no or few operations

Struct

Operations in wrong class

Just operate on the arguments of the method

Helper

Utility method

Util

A class should capture one and only one key abstraction

```
class Btree {

    public void printStudentsOnProbation() {
        blah
    }
```

What is the abstraction?

Duh

Duh

Comments that repeat the obvious in the code

```
// Check if student exists
if (studentExits(newStudent) ) {
    blah
    ...
```

Names

```
public class BTree {
        private Node root;
        private int order;

        /**
         * User method to insert into BTree
         * @param student
         */
        public void insert(Student student)
```

-1 name - use Java name

-1 return type -1

public Student searchForIndex(int index) throws IndexOutOfBoundsException

-1 name, Java names

```
class BTree(object):

    """
    find the leaf node where the value should be added
    """
    def insert(self, val):
    -1 use Python name
        node, slot = self.root.search(val)
        return node.insert(self, val, slot, None)


    """
    Given k, find the kth element from the Tree in lexicographical order
    """
    def findk(self,k):
    -1 use Python name
        output = []
```

```
public class BTree {
        private Node root;
        private int order;


        public List<Student> getStudentsGPALessThan(double gPA)
        {
-2 not part of Btree
```

Struct-10

```
class Node{
        //2 keys inside the node of order 3
        Key key1 = new Key();                   names -1 what is key1
        Key key2 = new Key();                    -1
        //3 children of an order 3              Duh comment
        Node node1;                             -1
        Node node2;                             -1
        Node node3;                             -1
        //store parent of node to traverse back DUH
        Node parent;
        //is leaf                               DUH Comments -6
        Boolean isLeaf;
        //default constructor
        Node(){}
        //Constructor of Node with 2 keys
        Node(Key key1, Key key2){
                this.key1 = key1;
                this.key2 = key2;
        }
}
```

II

```
class Node{
        Key left = new Key();
        Key right = new Key();

        Node leftChild;
        Node middleChid;
        Node rightChild;

        Node parent;

        Boolean isLeaf;

        Node(){}

        Node(Key key1, Key key2){
                left = key1;
                right = key2;
        }
}
```

```
public class BTree {
	private Node root;
	private int order;

	boolean nodeHasSpace(Node currentNode) {
-2 Util
		if(  (currentNode.key1 !=null) && ((currentNode.key1 !=null) || (currentNode.key2 !=null))  )
		{
			return true;
		}
		else
		{
			return false;
		}
}
```

```
class Node{

        boolean hasSpace(Node currentNode) {
                if(  (currentNode.key1 !=null) && ((currentNode.key1 !=null) || (currentNode.key2 !=null))
                {
                        return true;
                }
                else
                {
                        return false;
                }
         }
}
```

```
class Node{

	boolean hasSpace(Node currentNode) {
		return (currentNode.key1 !=null) && ((currentNode.key1 !=null) || (currentNode.key2 !=null)  )
	}
}
```

```
 class Node{

	boolean hasSpace(Node currentNode) {
		return currentNode.key1 !=null
	}
}
```

```
public class BTreeNode implements Comparable<BTreeNode> {
    //Each Node in BTree has list of keys                    -6 Duh comments
    private List<Student> keys = new ArrayList<>();
    //Each Node in BTree has list of children nodes
    private List<BTreeNode> children = new ArrayList<>();
    //parentNode stores the parent node for the current node
    private BTreeNode parentNode;
```

```
// assigning new root to an old root
root = newRoot;
```

```
public class BTree {
    //order of the BTree is pre-defined
    private final int order = 3;
    //The maximum keys a node can have in a BTree are order-1
    private final int maxKeys = order - 1;
    //root stores root node of the BTree
    BTreeNode root;
```

-6 Information hiding

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

    public void setRoot(BTreeNode root) {
        this.root = root;
    }
```

```
class BTreeNode:
-10 Struct
    def __init__(self, leaf=True):
        self.is_leaf = leaf
        self.parent = None
        self.keys = []
        self.child = []
        self.numkeys = 1

    def __str__(self):
        return f"** Each is a {self.keys} + {self.child} + leaf {self.is_leaf} + parent {self.parent} + keys
{self.numkeys} ******************"
```

Clients add elements (data) not nodes to B-Tree

Remove commented out code

```
class BTree:
    def __init__(self):
        self.root = None
        self.max_keys = 2
        self.split_index = 1

    # Insert node
    def insertElement(self, insertedValue):
        # print(f"^^^^^^^^^^^^^^^^^^^^^^^^STARTING TO Inserting {insertedValue}")

        if self.root == None:
            self.root = BTreeNode()
            # print("First element")
            self.root.keys.append(insertedValue)
        else:
            # Preemptive Split
            # if self.root.numkeys == self.max_keys:
            #     print(f"{self.root.numkeys} has reached MAX so splitting")
            #     self.split(self.root)
            # print(f"Inserting {insertedValue}")
            self.insertNew(self.root, insertedValue)
            # self.insert(self.root, insertedValue)
```

```
class BTree:
    def __init__(self):
        self.root = None
        self.max_keys = 2
        self.split_index = 1

    # Insert node
    def insertElement(self, insertedValue):

        if self.root == None:
            self.root = BTreeNode()
            self.root.keys.append(insertedValue)
        else:
            self.insertNew(self.root, insertedValue)
```

```
    def insertElement(self, insertedValue):

        if self.root == None:
            self.root = BTreeNode(insertedValue)
        else:
            self.insertNew(self.root, insertedValue)
```

```
public void print(String prefix) {
-4 don't print from Node class
  System.out.print(prefix);
  for (int i = 0; i < this.noOfElementsInNode; i++) {
    if (i != this.noOfElementsInNode - 1) {
      System.out.print(" | " + values.get(i));
    } else {
      System.out.print(" | " + values.get(i) + " | ");
    }
  }
  System.out.println();
  for (int i = 0; i < this.noOfChildNodes; i++) {
    //add tab for each level going down
    children.get(i).print(prefix + "\t");
  }
 }
```

Collection classes do not print

```
public class BTree<T extends Comparable<T>> implements Tree<T> {

  /**
   * this function adds a value and balances the BTree
   *
   * @param value adds a value of type {@link T} to the BTree.
   */
  @Override
   public void add(T value) {
-1 wrong return type
```

What type does Java's add method return?

```
public class BTree {
  private BTree() {... }
  private void findNode(Node insertedNode, String studentName) {...}
  private void insertStudent(String studentName, int redId, double gpa) {... }
  private void insertNode(final String studentName, int redId, double gpa) {...  }
  private void splitNode(Node parentNode, int position, Node leftNode) {...}
  private void showElement(int position) {...}
  private void checkIndex(int position) {... }
  private void showElement(Node node, int kPosition, int count) {... }
  private void showProbation() {...}
  private void showProbation(Node node) {... }
  private void showGPA4() {... }
  private void showGPA4(Node node) { }

public static void main(String[] args) {
  BTree btree = new BTree();
  btree.insertNode("mark", 1456, 4);
  btree.insertNode("adam", 1458, 4);
  btree.insertNode("aaron", 1459, 4);
  btree.insertNode("martyn", 1457, 2.5);
  btree.insertNode("venus", 1461, 1.5);
  btree.showElement(2);
  btree.showProbation();
  btree.showGPA4();
}
```

Can not use this class

public Student findKthStudent(int k) {

What does your languages library call this function?

```
public class Btree {
    int order;
    Bnode root;

    public void insert(Student student, Btree tree) {
        Bnode root = tree.root;
```

Why am I passing a BTree into a BTree object

```
public class Btree {

    public void Student findKthElement(Bnode root, int kthIndex) {
        Bnode root = tree.root;
```

So we need to access a node in the tree to be able to get the k'th element

BTree all ready has node so it should access the node, not make us do it.

Improper Inheritance

```
public class BTree extends Node {
    private Node root;
    ...
}
```

A BTree has a node which implies composition

It has a field of type Node

For a BTree to subclass Node it should be a type of a Node.

Misleading Name

```
class Btree
    def print_probation(self, target_gpa):
        probation = [ ]

        bunch of code to fill in probation, none prints

        return probation
```

```
class Btree

    def __traverse_to_insert(self, element, curr_node):          # line 110

        a really a lot of code

        return                                                   # line 232
```

```
class Btree

    def b_tree_search(self, data, node):
```

We know it is a B_tree search as it is in the BTree class
Why do we need a node?
What is the Python method name for search?

```
class Btree

    def add_to_list(self, current_student, student_list, gpa, k=float("inf")):
        if not k == float("inf):
            student_list.append(current_student)
        elif gpa == 4.0:
            if current_student.gpa >= 4.0:
                student_list.append(current_student)
        else:
            if current_student.gpa < gpa:
                student_list.append(current_student)
        return student_list
```

Why is this public?
What is k
Why use inf as a flag condition?

```
public void insert(Student s)
    code to check for empty tree not shown

     Node traversal = this.root;
     while (traversal != null) {
         if (traversal.children == 0) {
             traversal.addKeys(s);
             if (traversal.keys > order - 1) {
                 //split code here
                 splitNode(traversal);
             }
             break;
         } else {
             if (traversal.values[0].getStudentName().compareTo(s.getStudentName()) > 0) {
                 //left
                 traversal = traversal.pointers[0];
```

```
if (traversal.isLeaf()) {
    traversal.addKey(s);
    if (traversal.isFull()) {
        //split code here
        splitNode(traversal);
    }
    return;
```

```
public class BTree {
    private Node root;
    private in order = 3;
```

```
    // median
    traversal = traversal.pointers[1]
```

Only median if order is 3

So order is hard coded

Why create field for order?

In BTree class

```
public ArrayList<Student> findStudentsWithPerfectGPA() {
    ArrayList<Student> listOfStudent = new ArrayList<>();
    findStudentsWithPerfectGPA(root, listOfStudent);
    return listOfStudent;
}
```

-2 not part of BTree abstraction

```
private void findStudentsWithPerfectGPA(Node currentNode, ArrayList<Student> listOfStudent) {
```

-2 util

In BTree class

```
    public void insert(Student s) {
    -1 name - use Java name
    -1 return type
    -1 s : poor name
//        If the root is NULL then just insert the Student object with the help of addKey function
        if (this.root == null) {
            this.root = new Node(null);
            this.root.addKey(s);
        } else {
            Node traversal = this.root;
```

```
public bool add(Student newElement) {
   if (this.root == null) {
        this.root = new Node(newElement);
        return true;
    }
    Node traversal = this.root;
```

```
public Student put(String key, Student value) {
   if (this.root == null) {
        this.root = new Node(value);
        return true;
    }
    Node traversal = this.root;
```

```
public class Student {
        public void printRedIdIfOnProbation() {
                if(this.gpa < 2.85f) {
                        System.out.println(this.redId);
                }
        }
```

```
public class Student {
        public boolean onProbation() {
                return (this.gpa < 2.85f);
        }
```

```
public class BTree {

        /**
         * Root node of the B-Tree
         */
        private Node root;
        /**
         * Total number of keys in the B-Tree
         */
        private int totalKeys;

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

Comments repeat code

getRoot

No Information hiding

Why Public?

```
public void insertAtLeaf(Student studentToBeInserted, Node insertionNode) {
        Student[] keys = insertionNode.getKeys();
        if (insertionNode.getKeyCount() == 0)
                insertionNode.addKeyAtIndex(studentToBeInserted, 0);                /*first insertion at leaf*/
        else {// keyCount is 1 or 2
                if (keys[0].isStudentGreater(studentToBeInserted)) {
                        insertionNode.addKeyAtIndex(keys[0], 1);                    /*sort the nodes before
inserting at leaf*/
                        insertionNode.addKeyAtIndex(studentToBeInserted, 0);
                } else { // insert data directly if it's first insertion
                        insertionNode.addKeyAtIndex(studentToBeInserted, 1);
                }
        }
        insertionNode.setKeyCount(insertionNode.getKeyCount() + 1);
        totalKeys++;
}
```

```
/**
 * This class is used to implement sone helper functions to simplify code
 */
public final class HelperClass {
...

}
```

It contains one method

Not Fields

```
Public class BTree {
    private Node root;
    private Student fetchedStudent;
    MainMenu mmObj = new MainMenu;
```

```
public void traverse() {
    studentCount = 0;
    fetchedStudent = null;
    if (!isEmpty()) {
        traverse(root);
    } else {
        System.out.print("The tree is empty...");
    }
}
```

Who are you talking to?

Can not access result

```
public interface BTreeWithOrder3 {
        // root element of the tree
        public Node root = new Node();
        public int size = 0;

        // inserts student elements
        public void insert(Student studentElement);

        // prints details of Kth element from the tree
        public void getKthElementInTree(int k);

        // prints redId of probation students
        public void printProbationStudents();

        // prints student names with GPA 4.0
        public void printStudentsWithGPA4();

        // returns the size of the tree
        public int getSize();
}
```

How many implementations of Btree order 3 will you have

```
public class Utility {
	private int kCounter; // counter variable used to read Kth element in the list
	private static Student kthElement; // Kth element in the list is assigned to this variable

	public static void getKtheElementHelper(Node root, int k) {
		Utility utility = new Utility();
		utility.readKthElementInTree(root, k);
		if (kthElement != null) {
			System.out.println(k + " element in the list details are:\nname: " + kthElement.name +
					"\nredId: " + kthElement.redId + "\ngpa: " + kthElement.gpa);
		}
}
```

```
public class BufferNode extends Node {
    protected Student student3;
    protected Node middleChildNode2;

    public BufferNode(Student student1, Student student2, Student student3) {
        super(student1, student2);
        this.student3 = student3;
    }
}
```

What are:

middleChildNode2

student3

```
public ArrayList < Student > getList() {
-1 use java name
    createList(root);
    return allStudents;
}
```

```
public List < Student > asList() {
    createList(root);
    return allStudents;
}
```

how does this work?

createList stores the result the field allStudents

```
 public List < Student > asList() {
     return createList(root);
 }
```

don't use fields as temporary storage

```
BTree bTreeObj = new BTree();

// data aded to tree

bTreeObj.root.traverseFrontToBackInLexiOrder();
-6 information hiding,
need access to internal data structure
```

```
public class BTree {
    /**
     * The Order. // Duh
     */
    private int order;
    /**
     * The Root. //Duh
     */
    protected Node root;
    /**
     * The Max key size. Duh -6 Duh comments
     */
    private int maxAllowedKeyStudentSize;

    /**
     * It keeps student inserted in node counter
     */
    private int studentCounter;
```

```
public class BTree {
        String searchByK(int value) {
            if (size == 0 ) {
                return "Btree is empty";
            }
            return "The Node at index " + value + " contains " + searchableStudents.get(value);
```

Code is not talking to a person

```
 public class BTree {
         Student searchByK(int value) {
             if (size == 0 ) {
                 return null;
             }
             return searchableStudents.get(value);


public class BTree {
        Student get(int value) {
            if (size == 0 ) {return null;}
            return searchableStudents.get(value);
```

```
class Btree {
    private in order;
    private Node node;
    private Map<String, Student> mapStudentInfo
    private List<Student> studentData;

    public class Node {
        int numofKeysOccupied;
        String key[ ] = new String[ 2* order -1];
        Node child[ ] = new Node[2 * order];
        boolean leaf = true;
    }

    private class Student {
        String studentName;
        int studentRedId;
        double studentGPA;
```

What are

mapStudentInfo

studentData

Do we have 3 copies of the data?

Clients do not want to see Node

Clients do want to see Student

```
  public Node<T> getChildAtIndex(int index) {
      return children.get(index);
  }
```

We put data in trees

We expect to get data back, not nodes

Information hiding

- Physical
- Logical

Information Hiding

```
public class BTree {
    ...

    public Node getRoot() {
        return root;
    }

    public void setRoot(Node root) {
        this.root = root;
    }
```

Why would a client want to set the root?

```
public class BTree extends Comparable<T> {
    static private int OPTION1 = 1;
    static private int OPTION2 = 2;

    // B tree fields
    private int size;
    private int numElements
    private Node<T> root;

// fields to help with finding the 4th element
private int visitedIdx = 0;
private T lastVisited;
```

How does one compare BSTrees?

What is OPTION1 & OPTION2?

Don't need comment

    //B tree fields

What is the difference between

    size & numElements?

Last two fields are not part of the state of the object