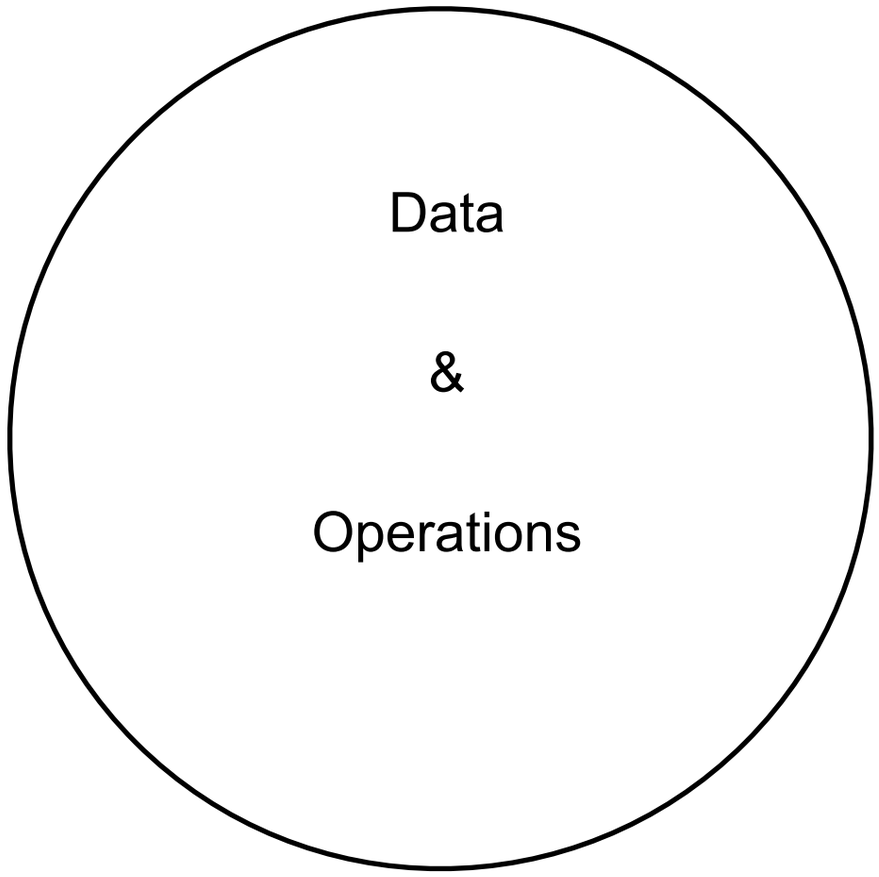
CS 635 Advanced Object-Oriented Design & Programming
Fall Semester, 2020
Doc 8 Assignment 1 Comments
Sep 15, 2020

Keep related data and behavior in one place



Struct -5 no methods

```
private class Node {
    public Node leftNode, rightNode;
    public int nodeValue;

    public Node(int nodeValue) {
        this.nodeValue = nodeValue;
        leftNode = null;
        rightNode = null;
    }
}
```

In MinHeap class

Utility Method -1

```
private int getHeight(Node rootNode) {
    while (rootNode != null) {
        int leftHeight = getHeight(rootNode.leftNode);
        int rightHeight = getHeight(rootNode.rightNode);

        return Math.max(leftHeight, rightHeight) + 1;
    }
    return 0;
}
```

```
private class Node {
    public Node leftNode, rightNode;
    public int nodeValue;

    public Node(int nodeValue) {
        this.nodeValue = nodeValue;
        leftNode = null;
        rightNode = null;
    }

    int getLeftHeight() {
         if leftNode == null return 0;
         return leftNode.getHeight();
     }

    public int getHeight() {
         return Math.max(getLeftHeight(), getRightHeight()) + 1
}
```

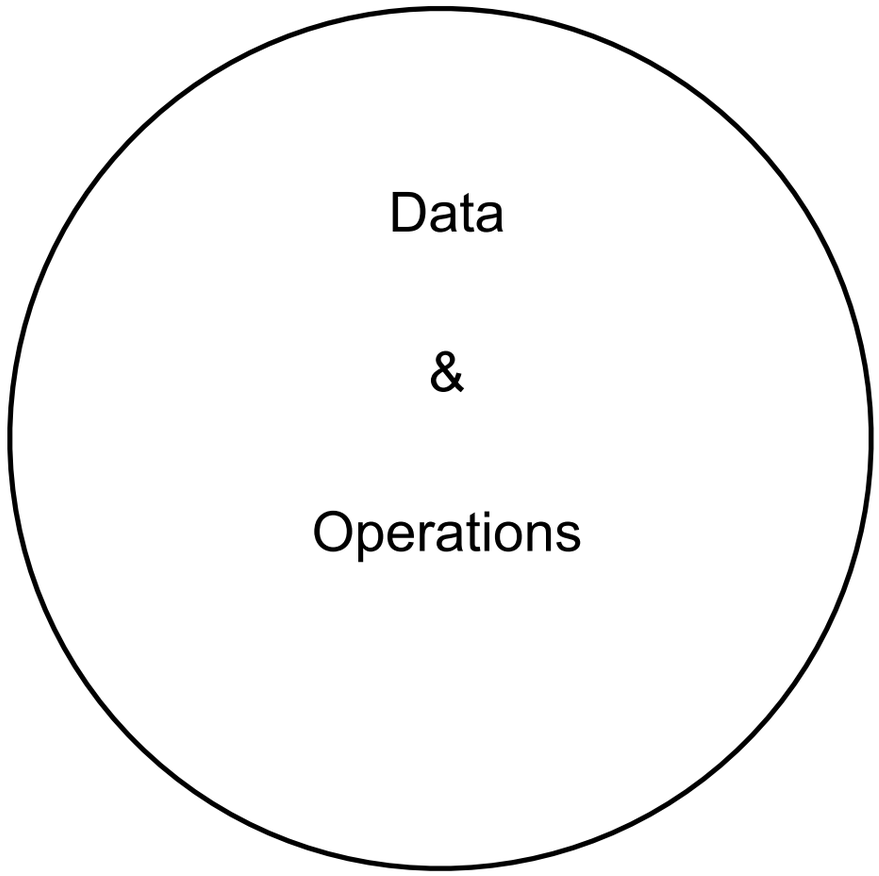
How many methods in your Heap class

Only use arguments

Arguments are nodes

Why aren't the methods in Node class

Keep related data and behavior in one place



Why is this important?
What benefits?

What are the benefits of OO?

Smaller pieces in natural location
More reusable code

Why the difference?

```
/**
 * This method is responsible for traversing the elements of the minheap in post order fashion
 *
 * @return list of elements in post order
 */
public List<Integer> traverseInPostOrder() {
```

```
    /**
     * This method returns a list of all the odd elements in the minheap while traversing in pre order fashion
     *
     * @return list of odd elements in pre order
     */
    public List<Integer> traverseInPreOrder() {
```

Why is Heap class dealing with odd values???

A class should capture one and only one key abstraction

What is the abstraction here?

How does odd values fit in?

```
    public void print() {
        //don't proceed if min-heap is empty
        comment not needed -1
        if (rootNode != null) {
            this.recursivePrint(rootNode);
        }
    }

    //recursively prints heap in postorder
Utility method -1
    private void recursivePrint(MinHeapNode currentNode) {
        if (currentNode.hasLeftChild()) {
            this.recursivePrint(currentNode.getLeftChild());
        }

        if (currentNode.hasRightChild()) {
            this.recursivePrint(currentNode.getRightChild());
        }
Don't use prints statements in classes -2
        System.out.println(currentNode.getValue());
    }
```

System.out.println, standard out

Mainly for debugging
Can not use the result

Name -1 print is not correct, method does not print
What is this comment? -1

```
        // Left side, right side, root
        public String printPostorder(){
                if(rootNode == null) return "";
                postOrder = "";
                // Call private helper class. Navigating down a tree is easier recursively
                if(rootNode.numberOfLeftChildren > 0){
                        printPostorder(rootNode.leftChild);
                }
                if(rootNode.numberOfRightChildren > 0){
                        printPostorder(rootNode.rightChild);
                }
                postOrder = postOrder.concat(Integer.toString(rootNode.nodeValue));
                return postOrder;
        }
```

```
public String postorder(){
      blah
}

public ArrayList postorder(){
      blah
}

public void postorder(Consumer<Integer> collector){
      blah
}
```

```
MinHeap example = new Heap();
...
Collection oddValues = new ArrayList();
oddValues.postOrder(element -> if (element % 2 != 0) oddValues.add(element))

oddValues.postOrder(element -> if (element % 2 != 0) System.out.print(element))
```

Info Hiding -5

```
public MinHeapNode getRootNode(){
    return rootNode;
}
```

Outside world now can access the nodes in the heap and modify them

Do you really need access to the nodes for testing?

```
public class MinHeap {
    private Node root;

Info Hiding? -5
    Node getRoot() {
        return root;
    }
```

```
    public void printOddValues() {
Duh comment -1
        //don't proceed if min-heap is empty
        if (rootNode != null) {
            this.recursivePrintOddValues(rootNode);
        }
    }
```

```
  public void printOddValues() {
     //don't proceed if min-heap is empty
      if (rootNode == null)  return;
      this.recursivePrintOddValues(rootNode);
  }
```

Use a guard to make it clear we don't proceed

```
  public void printOddValues() {
     if (isEmpty())  return;
     this.recursivePrintOddValues(rootNode);
  }

   public boolean isEmpty() {
      return rootNode == null;
   }
```

Make the code clearer so we don't need the comment

These comments are internal comments not public comments -1

```
    //calls private method to recursively iterate over heap
    public void print() {
        //don't proceed if min-heap is empty
        comment not needed -1
        if (rootNode != null) {
            this.recursivePrint(rootNode);
        }
    }
```

When we call your method we what to know what it does for us
Not how it is done

```
Duh  comment -1

 // Case there are no nodes, set the new node as root
 if(rootNode == null){
        rootNode = new Node(value);
        return value;
 }
```

Use Java names -1
What does Java return? -1

public int addValue(int value){

What name do Java collections use?
What do they return?
Why is that important?

Java API is huge, using common names makes it easier to use

Polymorphism

Allows collection classes to be used interchangeably

Why doesn't your Heap class implement the Collection interface?

```
while(!isInserted){
        // Swap values of the node if value to be inserted is less than the current "root"
        // Smallest value must always be the root
        if(currentRootNode.nodeValue > toBeInserted.nodeValue){
            int swapNodeValue = currentRootNode.nodeValue;
            currentRootNode.nodeValue = toBeInserted.nodeValue;
            toBeInserted.nodeValue = swapNodeValue;
        }
        // Case: currentNode has no children, favor left
        if(currentRootNode.numberOfLeftChildren == 0){
            currentRootNode.numberOfLeftChildren++;
            currentRootNode.leftChild = toBeInserted;
            isInserted = true;
        }
        // Case: currentNode has leftChild but no rightChild
        else if(currentRootNode.numberOfRightChildren == 0){
            currentRootNode.numberOfRightChildren++;
            currentRootNode.rightChild = toBeInserted;
            isInserted = true;
        }
        // Case: equal number of nodes on either side, favor left
```

```
if(currentRootNode.nodeValue > toBeInserted.nodeValue){
        int swapNodeValue = currentRootNode.nodeValue;
        currentRootNode.nodeValue = toBeInserted.nodeValue;
        toBeInserted.nodeValue = swapNodeValue;
}
```

```
if(currentRootNode.nodeValue > toBeInserted.nodeValue){
        currentRootNode.swapValueWith(toBeInserted);
}
```

```
// Case: currentNode has no children, favor left
if(currentRootNode.numberOfLeftChildren == 0){
        currentRootNode.numberOfLeftChildren++;
        currentRootNode.leftChild = toBeInserted;
        isInserted = true;
}
```

```
// Case: currentNode has no children, favor left
if(currentRootNode.numberOfLeftChildren == 0){
        currentRootNode.addLeftChild(toBeInserted);
        return toBeInserted.value;
}
```

```
public class MinHeap {
not part of state, just compute them when needed -2
    private final List<Integer> postOrderResult = new ArrayList<>();//store postorder traverse result f
    private final List<Integer> preOrderOddResult = new ArrayList<>();//store odd value preorder tra
testing
```

This needs a comment -1

```
public Node(int value, int height, Node parent) {
        // TODO Auto-generated constructor stub - remove these comments -1
        this.value = value;
        this.height = height;
        this.parent = parent;
        compare(this.parent, this);
        if(this.parent!=null)
                parent.increaseHeight();
}
```

-1 misleading name

-1 Utility(Helper method)

Yet another method for Node class

```
private void compare(Node parent, Node child) {
        // TODO Auto-generated method stub
        while (parent != null) {
                if (parent.getValue() > child.getValue()) {
                        int temp = parent.getValue();
                        parent.setValue(child.getValue());
                        child.setValue(temp);
                }
                parent = parent.getParent();
        }
}
```

```
    private void compare(Node parent, Node child) {
        while (parent != null) {
            if (parent.getValue() > child.getValue()) {
                int temp = parent.getValue();
                parent.setValue(child.getValue());
                child.setValue(temp);
            }
            parent = parent.getParent();
        }
}
```

```
    private void compare(Node parent) {
        while (parent != null) {
            if (parent.getValue() > this.getValue()) {
                int temp = parent.getValue();
                parent.setValue(this.getValue());
                this.setValue(temp);
            }
            parent = parent.getParent();
        }
}
```

```
    private void insert(Node parent) {
        if (parent == null) return;
        if (parent.getValue() > this.getValue()) {
            this.swapValue(parent);
        }
        this.compare(parent.getParent());
}
```

```
    private void compare(Node parent) {
        while (parent != null) {
            if (parent.getValue() > this.getValue()) {
                this.swapValue(parent);
            }
            parent = parent.getParent();
        }
}
```

-1 comment not helpful

// Create empty heap
final MinHeap minHeap = new MinHeap();

Why are you doing this?

```
public static void main(final String[] args) {
    final MinHeap minHeap = new MinHeap();

    Scanner action = new Scanner(System.in);

    int selection = 0;
    // Take user input, handle is user enters non-integer and reprompt for input
    while (selection == 0) {
        try {
            System.out.println("Please enter a number to select an action: ");
            System.out.println("1) Add a value to the min-heap");
            System.out.println("2) Print the min-heap in postorder");
            System.out.println("3) Print all odd values in preorder");
            System.out.println("4) Quit");
            selection = action.nextInt();
        } catch (InputMismatchException e) {
            action.nextLine();
        }
}
```

etc

Your comment indicates what the name should be -1

```
    //returns true if the parent is greater than child.
    private boolean isSwapingRequired(Node parent,Node child) {
        if(parent.getData()<child.getData())
            return false;
        else
            return true;
```

```
public void addNode(Integer dataValue) {
    if (this.isEmpty())
        root = new Node(dataValue);
    else
        addHelper(dataValue, root);
}
```

Physical information hiding

Logical information hiding

We add elements to the heap, not nodes

Nodes are an internal (private) implement detail

Other implementations do not use nodes

addHelper

Name indicates you know it does not belong here

```
public class MinHeap {
    private Node root;
    private ArrayList<Integer> treeValues = new ArrayList<>();
```

What is treeValues?

```
public void printPostorderAll(HeapNode node)
{
      blah
        System.out.print(node.getValue());
        System.out.print(" ");
    }
}
```

```
// This function is for unit testing purpose only to validate the pre-order output using StringBuild
// Prints the odd nodes of the heap in Pre-order
public String unitTest_PreorderOdd(HeapNode node)
```

The only reason you need this method for testing is you are printing to standard out
So have no access to the result.