CS 635 Advanced Object-Oriented Design & Programming
Spring Semester, 2010
Doc 6 Assignment 1 Comments
Jan 11, 2010

Comments?

```
public class BSTNode {
    private String data;
    private BSTNode left;
    private BSTNode right;
    private String alphabeticEndingInNS;
    private String reverseOrder;
```

State Versus Local Variables & Parameters

State - characteristic that an object keeps as long as it exists

Part of abstraction
Used in multiple methods

Local variables
Used for calculations in a single method

Parameters
Provide data from/to methods

State definition from http://www.kiv.zcu.cz/~ledvina/vyuka/Java/javanotes/Notes/chap35/ch35_3.html

A verses B

A

```
public class Foo {
    public int a(int x) {
        return b(x) + 5;
    }

    private int b(int z) {
        return z *10;
    }
}
```

B

```
public class Foo {
    private int temp1;
    private int temp2;

    public int a(int x) {
        temp1 = x
        return temp2 + 5;
    }

    private void b(int z) {
        temp2 = temp1 *10;
    }
```

Computed Values and State

```
public class BinaryTree {
    private Node root;

    public int size() {
        return root.left().size() +
            root.right().size();
    }
}
```

```
public class BinaryTree {
    private Node root;
    private size;

    public int size() {
        return size;
    }

    public boolean add(String value) {
        boolean wasAdded = root.add(value);
        if (wasAdded) size++;
        return wasAdded;
    }
```

Comments?

```
public class BSTNode {
    public String data;
    public BSTNode left;
    public BSTNode right;
}
```

Comments?

```
public class BSTNode {
    private String data;
    private BSTNode left;
    private BSTNode right;

    public void setData(String value) { data = value;}
    public void setLeft(BSTNode value) { left = value;}
    public void setRight(BSTNode value) { right = value;}
    public String getData() { return data;}
    public BSTNode getLeft() { return left;}
    public BSTNode getRight() { return right;}
}
```

Helper Methods

```
public class BinaryTree {
    private Node root;

    public void add(String value) {
        if (root == null)
            root = new Node(value);
        else
            add(root, value);
    }
```

```
    private void add(Node aNode, String value) {
        if (aNode.value == value) return;
        if (aNode.value < value) {
            if (aNode.right == null){
                aNode.right = new Node(value);
                return;
                }
            return add(aNode.right, value)
        }
        if (aNode.value > value ) {
            etc.
    }
}
```

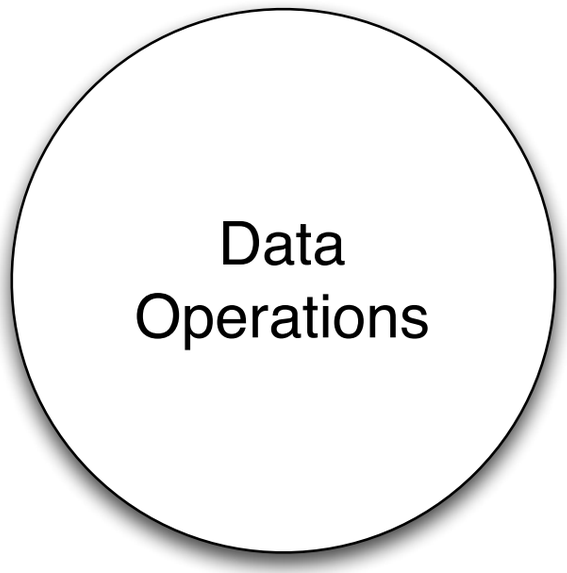
Helper Methods

Only operate on parameters
Do not access state of object

Can be made static

Functions in method clothing

Remember This?



Heuristics

Keep related data and behavior in one place

A class should capture one and only one key abstraction

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

What was the point of assignment 0?

Data Classes

Lots of sets and gets

If you have a data class

Look for helper methods in your code

Are the helper methods dealing with the data in your data class?

You just found operations for your data class

Keep data & operations together

Comments?

```
public class BinaryTree {
    private Node root;

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

Comments

```
public class BinaryTree {
    private Node root;

    public void addNode(Node aNode) {
        if (root == null)
            root = aNode;
        else
            etc
    }
```

Comments?

```
public class BinaryTree {
    private Node root;

    public boolean IsTreeEmpty() {
        if (root == null)
            return true;
        else
            return false;
    }
```

```
public class testBST extends TestCase {
```

Comments?

```
public class BinaryTree {
    private Node root;

    public void addData(String value) {
        if (root == null)
            root = new Node(value);
        else
            etc
    }
```

Java Standard

public boolean add(E e)
public void add(int index, E element)

public V put(K key, V value)

Why it Matters

Easier to remember standard names

Polymorphism

Comments?

```
public class BinaryTree {
    private Node root;

    public void printReverseAlphabeticOrder() {
        blah
        blah
        System.out.println(value);
        blah
        blah
}
```

Better but ...

```
public class BinaryTree {
    private Node root;

    public String reverseAlphabeticOrder() {
        String result = ''
        blah
        result += value;
        blah
        return result;
    }
```

Better still

```
public class BinaryTree {
    private Node root;

    public ArrayList reverseAlphabeticOrder() {
        ArrayList result = new ArrayList()
        blah
        result.add(value);
        blah
        return result;
        }
```

Abstraction

```
public class BinaryTree {
    private Node root;

    private static final Pattern ENDS_IN_OR_S = Pattern.compile(".*[ns]");
```

Duh Comments

```
public class BinaryTree {
    /* root of the tree */
    private Node root;

    /* constructor */
    public BinaryTree() {
    }

/* add string to tree */
public void add(String value) {
    blah
}
```

Static

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

    public void static addNode(Node aNode) {
        if (root == null)
            root = aNode;
        else
            etc
    }
```

Comments?

```
public class BinaryTree {
      private  Node root;
      private int treeSize;

      public int treeSize() {
                return treeSize;
      }
```

Comments?

```
public class BinaryTree {
    private  Node root;

    public void insertData(String key) {
        try {
            if (key == null )
                System.out.println("Null string");
            else if (key.trim().isEmpty())
                System.out.println("Empyt string");
            else {
                blah
                blah
            }
        } catch(Exception e ) {
            System.out.println("Bad string " + e.toString();
        }

}
```

Comments?

```
public class BinaryTree {
      private  Node root;
      private int treeSize;

      public void insertData(BinaryTree aTree) {
            aTree.add("a");
            aTree.add("b");
            aTree.add("c");
      }
```

Comments

```
public class TestTree {
    @Test
    public void someTest() {
        blah
        for (int k = 0; k < list.size(); k++) {
            if (!(list.get(k).equals(listEndsWithNS.get(i)))) {
                flag = false;
                break;
            }
            else
                flag = true;
        }
        assertTrue(flag);
    }
```

Better?

```
public class TestTree {
    @Test
    public void someTest() {
        blah
        allEqual = true;
        for (int k = 0; k < list.size(); k++) {
            if (!(list.get(k).equals(listEndsWithNS.get(i))))
                allEqual = false;
        }
        assertTrue(allEqual);
    }
```

How about this?

```
public class TestTree {
    @Test
    public void someTest() {
        blah
        for (int k = 0; k < list.size(); k++)
            assertEquals(list.get(k), listEndsWithNS.get(i));
    }
```

flag

Not the worst possible name - but close

What condition are you checking for?

Pick a name this indicates the condition

Comments

```
public class BinaryTree extends Node {
    private Node root;
    etc.
```

How is this an automated test?

```
public void testPrintSN {
    BinaryTree tree = new BinaryTree();
    tree.add("a");
    tree.add("n");
    tree.add("s");
    tree.add("b");
    tree.add("ns");
    tree.printSN();
}
```

Boolean anyone?

```
private String insertNode(Node aNode, String data) {
      blah
      blah
      if (aNode.value == data) return "failure";
      blah
      blah
      return "success";
}
```

A versus B

A

```
public boolean add(String data) {
    boolean result = root.add(data);
    return result;
}
```

B

```
public boolean add(String data) {
    return root.add(data);
}
```