Michael Brower & Vitor Cavalcante
11-12-19
CSCI 260

Programming Challenge:

```
import java.util.Scanner;


public class btree1
{
	public static void main(String[] args)
	{
		Scanner scan = new Scanner(System.in);


		/* Creating object of BT */
		BT bt = new BT();

		/* String st1 = "Vitor";
		String st2 = "David";

		System.out.println("The number comparison is: " + st1.compareTo(st2));

		*/



		/* Perform tree operations */
		System.out.println("Binary Tree Test\n");
		char ch;
		do
		{
			System.out.println("\nBinary Tree Operations\n");
			System.out.println("1. insert ");
			System.out.println("2. search");
			System.out.println("3. count nodes");
			System.out.println("4. check empty");
```

```
                        int choice = scan.nextInt();


                        switch (choice)
                        {
                        case 1 :
                                System.out.println("Enter String element to insert");
                                bt.insert( scan.next() );
                                break;
                        case 2 :
                                System.out.println("Enter String element to search");
                                System.out.println("Search result : "+ bt.search(
scan.nextLine() ));
                                break;
                        case 3 :
                                System.out.println("Nodes = "+ bt.countNodes());
                                break;
                        case 4 :
                                System.out.println("Empty status = "+ bt.isEmpty());
                                break;
                        default :
                                System.out.println("Wrong Entry \n ");
                                break;
                        }


                        /* Display tree */
                        System.out.print("\nPost order : ");
                        bt.postorder();
                        System.out.print("\nPre order : ");
                        bt.preorder();
                        System.out.print("\nIn order : ");
                        bt.inorder();
                        System.out.println("\n\nDo you want to continue (Type y or n)
\n");
                                ch = scan.next().charAt(0);
                        } while (ch == 'Y'|| ch == 'y');
                }
        }
```

```
import java.util.Scanner;


/* Class BTNode */
class BTNode
{
    BTNode left, right;
    String data;


    /* Constructor */
    public BTNode()
    {
        left = null;
        right = null;
        data = "Blank";
    }


    /* Constructor */
    public BTNode(String n)
    {
        left = null;
        right = null;
        data = n;
    }


    /* Function to set left node */
    public void setLeft(BTNode n)
    {
        left = n;
    }


    /* Function to set right node */
    public void setRight(BTNode n)
    {
```

```
        right = n;
    }

    /* Function to get left node */
    public BTNode getLeft()
    {
        return left;
    }

    /* Function to get right node */
    public BTNode getRight()
    {
        return right;
    }

    /* Function to set data to node */
    public void setData(String d)
    {
        data = d;
    }

    /* Function to get data from node */
    public String getData()
    {
        return data;
    }
}

/* Class BT */
class BT
```

```
{
    private BTNode root;


    /* Constructor */
    public BT()
    {
        root = null;
    }


    /* Function to check if tree is empty */
    public boolean isEmpty()
    {
        return root == null;
    }


    /* Functions to insert data */
    public void insert(String data)
    {
        root = insert(root, data);
    }


    /* Function to insert data recursively */ //Check if less or equal to or greater and organize the insert
    private BTNode insert(BTNode node, String data)
    {
        if (node == null)
            node = new BTNode(data);
        else
        {
            if (node.getRight() == null)
                node.right = insert(node.right, data);
            else
                node.left = insert(node.left, data);
        }
        return node;
```

```
}


/* Function to count number of nodes */
public int countNodes()
{
        return countNodes(root);
}


/* Function to count number of nodes recursively */
private int countNodes(BTNode r)
{
        if (r == null)
                return 0;
        else
        {
                int l = 1;
                l += countNodes(r.getLeft());
                l += countNodes(r.getRight());
                return l;
        }
}


/* Function to search for an element */
public boolean search(String val)
{
        return search(root, val);
}

/* Function to search for an element recursively */
private boolean search(BTNode r, String val)
{
        if (r.equals(val))
                return true;
        if (r.getLeft() != null)
                if (search(r.getLeft(), val))
                        return true;
```

```
        if (r.getRight() != null)
                if (search(r.getRight(), val))
                        return true;
        return false;
}


/* Function for inorder traversal */
public void inorder()
{
        inorder(root);
}
private void inorder(BTNode r)
{
        if (r != null)
        {
                inorder(r.getLeft());
                System.out.print(r.getData() +" ");
                inorder(r.getRight());
        }
}


/* Function for preorder traversal */
public void preorder()
{
        preorder(root);
}


private void preorder(BTNode r)
{
        if (r != null)
        {
                System.out.print(r.getData() +" ");
                preorder(r.getLeft());
                preorder(r.getRight());
        }
}
```

```
    /* Function for postorder traversal */
    public void postorder()
    {
        postorder(root);
    }

    private void postorder(BTNode r)
    {
        if (r != null)
        {
            postorder(r.getLeft());
            postorder(r.getRight());
            System.out.print(r.getData() +" ");
        }
    }

}

/* Class BinaryTree */
```