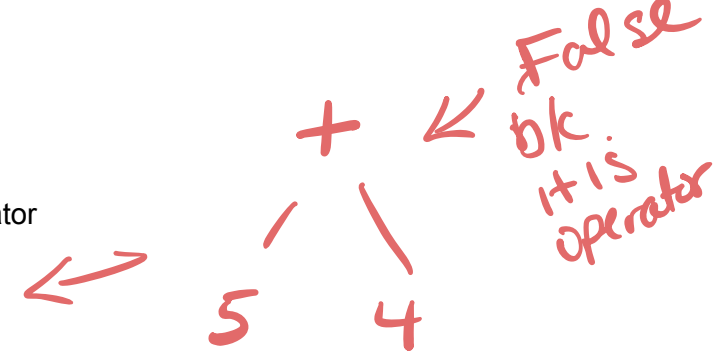
```
public class ExpressionTree{


   //You must write this method:
   //Calculate the value of the entire tree
   public double evaluate(){

     return 10000000000000.0;//replace this
   }

   //You must write this method:
   //Return a string representation of the tree
   //A value is just hte string of the value
   //An operation has parenthesis around
   //such as
   //"12.5"  //just a value
   //"(5.0-8.0)" //a tree with 2 value children
   //"(12.5*(5.0-8.0)) //a tree that is the product of the previous two example trees
   //"(((2.0+1.0)/(8.0*0.43)) - 1.0)" //a tree with more
   public String toString(){
    return "replace this with your code";
   }

 private double value;
 private ExpressionTree left,right;
 private char operator;

 //Tree can be a value
 public ExpressionTree(double val){
  value = val;
  left = null;
  right = null;
 }

 //Tree can be an operator that connects two sub-trees
 public ExpressionTree(char op, ExpressionTree l, ExpressionTree r){
   operator = op;
   left = l;
   right =r;
 }

 //Return true when the node is a value, false when it is an operator
 //when the children are null, the current tree is an operator
 private boolean isValue(){
   return left==null && right ==null;
 }

 //Return false when the node is a value, true when it is an operator
 private boolean isOperator(){
```

value
left
right
operator

Default

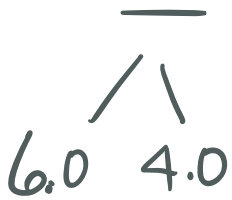


true

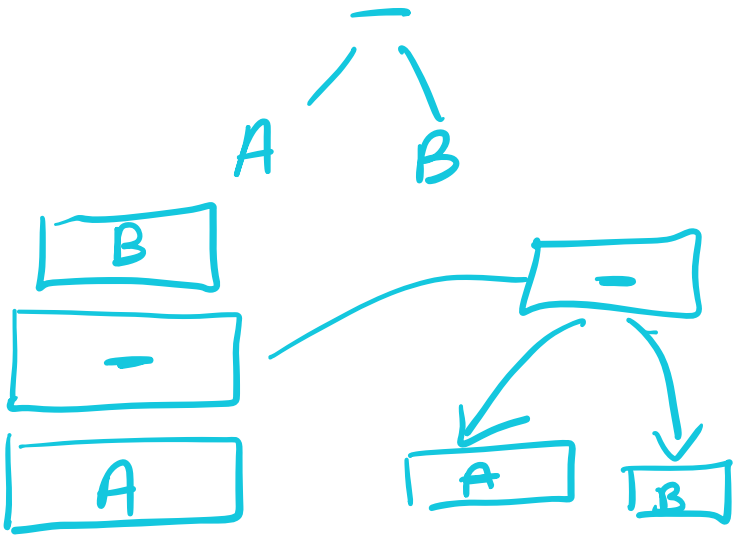
```
    return !isValue();
  }

  //To simplify things, you get a method to use to convert the
  //operator and two values into a result.
  private double apply(double a, double b, char op){
   if(op == '+'){
    return a+b;
   }else if(op == '-'){
    return a-b;
   }else if(op == '*'){
    return a*b;
   }else{ //  if(op == '/'){ //or any invalid operators
    return a/b;
   }

  }

 }

public class TreeDriver{
  public static void main(String[]args){

    //1. Test out single value trees. They are the base case and
    //should be tested first!
    //2. Uncomment the 'Trees with children' section
    //3. Uncomment the 'Multi level trees' so you can work with more complex examples..


    //1. Single value trees
    ExpressionTree one = new ExpressionTree(1.0);
    ExpressionTree two = new ExpressionTree(2.0);
    ExpressionTree three = new ExpressionTree(3.0);
    ExpressionTree four = new ExpressionTree(4.0);
    ExpressionTree five = new ExpressionTree(5.0);
    ExpressionTree ten = new ExpressionTree(10.0);


    //Check your toString for single value trees.
    System.out.println("Should print 1.0 2.0 3.0 4.0 5.0 10.0 on separate lines");
    System.out.println(one);
    System.out.println(two);
    System.out.println(three);
    System.out.println(four);
    System.out.println(five);
    System.out.println(ten);

    //check your evaluate:

    System.out.println("Should print true 6 times");
    System.out.println(one.evaluate()==1.0);
    System.out.println(two.evaluate()==2.0);
```

```
    System.out.println(three.evaluate()==3.0);
    System.out.println(four.evaluate()==4.0);
    System.out.println(five.evaluate()==5.0);
    System.out.println(ten.evaluate()==10.0);


    //2. Trees with children
    //These need to be evaluated:
    /**
    ExpressionTree a = new ExpressionTree('+',one,one);//2.0  "(1.0 + 1.0)"
    ExpressionTree b = new ExpressionTree('-',five,ten);//-5.0 "(5.0 - 10.0)"
    ExpressionTree c = new ExpressionTree('*',three,four);//12.0 "(3.0 * 4.0)"
    ExpressionTree d = new ExpressionTree('/',one,two);//0.5 "(1.0 / 2.0)"
    System.out.println("Should print out the same thing as the comments in the source code:");
    System.out.println(a.toString() + " is equal to " + a.evaluate());
    System.out.println(b.toString() + " is equal to " + b.evaluate());
    System.out.println(c.toString() + " is equal to " + c.evaluate());
    System.out.println(d.toString() + " is equal to " + d.evaluate());
    */
    /** output:
    (1.0+1.0) is equal to 2.0
    (5.0-10.0) is equal to -5.0
    (3.0*4.0) is equal to 12.0
    (1.0/2.0) is equal to 0.5
    */


    //3. Multi level trees which need to be evaluated:
    /**
    ExpressionTree e = new ExpressionTree('/',c,two);
    ExpressionTree f = new ExpressionTree('-',c,ten);
    ExpressionTree g = new ExpressionTree('+',b,c);
    ExpressionTree h = new ExpressionTree('*',a,d);
    ExpressionTree i = new ExpressionTree('+',h,one);
    System.out.println("Should print out the same thing as the comments in the source code:");
    System.out.println(e.toString() + " is equal to " + e.evaluate());
    System.out.println(f.toString() + " is equal to " + f.evaluate());
    System.out.println(g.toString() + " is equal to " + g.evaluate());
    System.out.println(h.toString() + " is equal to " + h.evaluate());
    System.out.println(i.toString() + " is equal to " + i.evaluate());
    */
    /** output:
    ((3.0*4.0)/2.0) is equal to 6.0
    ((3.0*4.0)-10.0) is equal to 2.0
    ((5.0-10.0)+(3.0*4.0)) is equal to 7.0
    ((1.0+1.0)*(1.0/2.0)) is equal to 1.0
    (((1.0+1.0)*(1.0/2.0))+1.0) is equal to 2.0
    */

  }
}
```

((6.0 - 4.0)



(A - B)



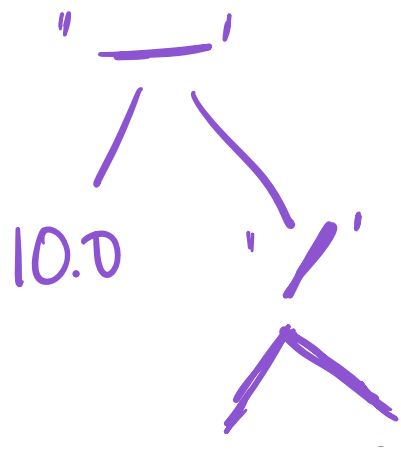

```
if root == null
    return;
if it is an operator (isOperator)
    print the operator.
    with a (
```

"(10-(1.0/3.5))"

- if single node tree
  print value

```
if (root == null)
    return value
```

else
  if isOperator is true
    print "("
  go to left node
    print value
  go back to root
    and print operator
  then go to right node



1.0 3.5

if is operator is true
print '('
Go to left
print value
go back to root