**Question 6:** (25 points) In class we discussed some code that created and manipulated computation trees. The tree is composed of nodes that are subtypes of Computation: AddExpr, SubExpr, MultExpr, and DivExpr, Constant, and Decorator (whose definition is shown below).

abstract public class Computation {
    protected final Computation left, right;
    public void accept(CompVisitor v){};

    // Binary op constructors: AddExpr(), SubExpr(), MultExpr(), and DivExpr()
    Computation (Computation l, Computation r) {
        left = l;  right = r;
    }

    // Constant and Decorator constructor
    Computation () {left = right = null;}
}

To manipulate Computation's we created an interface called CompVisitor (also shown below) that represents the Visitor Pattern. We also examined several classes that implemented the CompVisitor interface.

interface CompVisitor {
    public void visit(Constant c);
    public void visit(AddExpr c);
    public void visit(SubExpr c);
    public void visit(MultExpr c);
    public void visit(DivExpr c);
    public void visit(Decorator c);
}

Write a class called DepthVisitor that implements CompVisitor. This class will be used to calculate the depth of an expression (which is represented by a computation tree). Abstractly, the depth of an expression is defined as:

\[
\text{depth}(\text{int}) = 0,
\]

\[
\text{depth}(e1 \text{ op } e2) = 1 + \max(\text{depth}(e1), \text{depth}(e2))
\]

Justify your handling of Decorators.
**Sample Answer**

Decorators are transparent wrappers. They don't change the underlying expression. Therefore, they shouldn't add to the depth of the expression.

```java
public class DepthVisitor implements CompVisitor {
    private int result = 0;

    public int getResult() {
        return result;
    }

    private int doLeft(Computation c) {
        c.left.accept(this);
        return result;
    }

    private int doRight(Computation c) {
        c.right.accept(this);
        return result;
    }

    public void visit(AddExpr a) {
        result = 1 + java.lang.Math.max(doLeft(a), doRight(a));
    }

    public void visit(Constant a) {
        result = 0;
    }

    public void visit(Decorator a) {
        a.decoratee.accept(this);
    }

    public void visit(DivExpr a) {
        result = 1 + java.lang.Math.max(doLeft(a), doRight(a));
    }

    public void visit(MultExpr a) {
        result = 1 + java.lang.Math.max(doLeft(a), doRight(a));
    }

    public void visit(SubExpr a) {
        result = 1 + java.lang.Math.max(doLeft(a), doRight(a));
    }
}
```